*Ad Hoc & Sensor Wireless Networks* Vol. 00, pp. 1–22
Reprints available directly from the publisher
Photocopying permitted by license only

©2009 Old City Publishing, Inc.
Published by license under the OCP Science imprint,
a member of the Old City Publishing Group

# MLMAC – An Adaptive TDMA MAC Protocol for Mobile Wireless Sensor Networks

STEPHAN MANK, REINHARDT KARNAPKE AND JOERG NOLTE

*Distributed Systems and Operating Systems Group,*
*Brandenburg University of Technology Cottbus, Cottbus, Germany*
*E-mail: {smank, karnapke, jon}@informatik.tu-cottbus.de*

*Received: December 15, 2007. Accepted: December 24, 2008.*

Today, wireless sensor networks are typically realized using cheap radio transceivers offering low bandwidth communication only. When physical events in the real world trigger spontaneous communication in many nodes, the single communication channel is under heavy load and many messages are lost due to collisions. CSMA/CA schemes are well suited to spontaneous communication, but do not provide a high channel utilization under heavy load. TDMA protocols have some conceptual advantage here, but in the case of mobile sensor nodes they need to be adaptive and establish TDMA schedules on demand. In this paper we introduce the MLMAC (mobile LMAC) protocol. MLMAC is a novel TDMA based MAC protocol that can react on changing radio neighborhoods in mobile networks. In addition to the conceptual description of MLMAC, we also present the results of real experiments with a group of mobile sensor nodes based on RCX robots.

*Keywords:*

## 1 INTRODUCTION

Sensor networks are collections of small sensor nodes with wireless neighborhood broadcast facilities. Since sensor networks shall be deployed in large scales (possibly thousands of nodes [1,2]), today's sensor nodes are typically optimized for minimum cost. This low cost policy often results in the usage of cheap radio transceivers offering low bandwidth communication only. These lack most of the common capabilities of WLAN or bluetooth networks. Even typical tasks like medium access control or the addressing of individual

nodes in the direct radio neighborhood are left entirely to software layers [3], no hardware support is supplied.

In this paper we present Mobile LMAC (MLMAC), a novel TDMA MAC protocol for mobile wireless sensor networks. It is a modification of the lightweight medium access protocol (LMAC) [4] which has been adopted for our RCX robots and for a different scenario. The mobility of nodes in our scenario is the reason why finding a slot for transmission is more difficult and must be repeated as nodes can enter or leave a radio neighborhood at any time. To the best of our knowledge, no previous implementation of LMAC has been used on real hardware.

This paper is structured as follows: Section 2 describes our MAC protocol while Section 3 shows some aspects of the implementation. Section 4 describes the setup of our experiments and their results. We finish with related work in Section 5 and a conclusion in Section 6.

## 2 MOBILE LMAC (MLMAC)

### 2.1 LMAC

In [4] L. F. W. van Hoesel and P. J. M. Havinga introduced LMAC (lightweight medium access control), a MAC protocol for stationary sensor networks with a single sink. LMAC is based on a TDMA scheme. Time is divided into frames and slots. Each node reserves a slot in which it can send, this slot reoccurs every frame. Every Slot is further divided into a phase used to send a control message and another one where the data payload is transmitted.

Table 1 shows the contents of a LMAC control message. Its total size amounts to 12 Bytes. It contains the identity of the sender and its slot number followed by the most important field `Occupied Slots`, which represents a bitmask of Slots. An unused slot is represented by a 0 while a 1 represents an occupied one. Thus it is possible for every node to determine unoccupied slots

| Description | Size (bytes) |
|---|---|
| Identification | 2 |
| Current Slot Number | 1 |
| Occupied Slots | 4 |
| Distance to Gateway | 1 |
| Collision in Slot | 1 |
| Destination ID | 2 |
| Data Size (bytes) | 1 |
| Total | 12 |

TABLE 1
The control message used in LMAC

by combining the control messages of its neighbors. This is done by performing a simple OR operation on the fields `Occupied Slots` of all received control messages. The distance to the Gateway is also transmitted, along with information of overheard collisions. Finally, the ID of the destination (used for power management) and the size of the data unit are given.

The initialization of nodes is started by the gateway, which defines its own slot and is used for synchronization. After one frame, all direct neighbors of the gateway know its slot and choose their own ones. This information is transmitted to their neighbors who synchronize on these messages. After each frame, a new set of nodes with a higher hop distance from the gateway is synchronized until every node knows its slot. These slots only need to be locally unique, as the nodes only compete with others up to 2 hops distant. To conserve node energy, a node's transceiver is turned off for the remainder of the current slot when it is not addressed in the control message.

### 2.2 Differences and similarities between LMAC and MLMAC

All of the attributes described above make LMAC an excellent choice for its intended scenario, a stationary sensor network with communication only from the nodes to the sink. But our scenario is quite different. We assume a mobile sensor network, consisting, e.g., of robots equipped with fire extinguishers. Those robots would patrol an endangered region, and act on threats that arise. If one of them found a fire, it would call in others for support, in addition to informing the next fire engine. Another application would be a toxic waste dump, where leakage of certain gases might be detected. In these cases, it is often to expensive or not feasible to permanently position sensor nodes with possibly expensive sensors on every possible location. Therefore, mobility is needed.

The choice of such scenarios has a number of consequences for the developed MAC protocol. First, there may not be a gateway to start the synchronization (see Section 2.2.1). Second, the chosen slots are not fixed in time. Due to mobility, it may become necessary for a node to choose a new slot when it enters a different radio neighborhood (see Section 2.2.2). Third, MLMAC needs to differentiate between bidirectional and unidirectional links (Section 2.3).

#### 2.2.1 Slot synchronization

In LMAC the synchronization is always started by the gateway. As MLMAC was developed for mobile networks which can possibly contain multiple sinks, this is not possible. The node that wants to send a packet first starts the synchronization. This removed the necessity to use the field `Distance to Gateway` for synchronization. Even when it is not used for synchronization, the field `Distance to Gateway` could be used to support routing decisions in stationary sensor networks. In mobile sensor networks however, this distance could not be determined only once and saved for further

| Description | Size (bytes) |
|---|---|
| Identification | 1 |
| Slot number and Status | 1 |
| Occupied Slots | 1–2 |
| Identity of the Synchronization | 1 |
| Age of Synchronization | 1 |
| Total | 5–6 |

TABLE 2
The control message used in Mobile LMAC (MLMAC)

use, as the mobility of node leads to a change in topology after a certain time. This time depends on the range of the transceivers and on the speed of the nodes, but eventually the change in topology will take place. The field `Distance to Gateway` is removed from the header of MLMAC. The fields `Destination ID` and `Collision in Slot` were not used either, because of the used hardware. There was no way to shut down the transceivers and the radio module used a built in checksum to discard faulty packets. Thus, collisions could not be detected directly and this part of LMAC was not implemented. As this decision is based solely on the used radio modules, it could be revoked in the future, when a different platform is used (see Section 6).

MLMACs control message format is shown in Table 2. This control message is quite different from the one used in LMAC. Due to our small sensor network, we reduced the field containing the identity of the sender to one Byte. `Slot number and Status` contains 5 Bit for the senders slot and 3 Bit for its status. The field `Occupied Slots` is used exactly as in LMAC, only its size is reduced.

The field `Occupied Slots` in the control message of a node contains the used slots of all its neighbors and itself. In the case of node 4 on Figure 1 the slots 3, 4, 5, 6 and 8 would be marked as used, which results in a representation as 00111101. Note that in this example the third bit from the left represents the third slot.

The figure shows how slots are chosen with a simple example containing only eight nodes. In this example you can see that node 2 is not synchronized yet. It receives the control messages from its neighbors and combines them. 10000100 (from node 1) — 00111000 (from node 3) — 00111101 (from node 4) = 10111101 (seen on node 2).

This means that node 2 can choose between slots 2 and 7. If it chooses slot 2, its control message would contain the bitmask 11110000 in the field `Occupied Slots`, as node 2 receives messages from nodes 1, 3, and 4 and adds its own choice. If it chooses slot number 7 the field `Occupied Slots` would contain the bitmask 10110010.

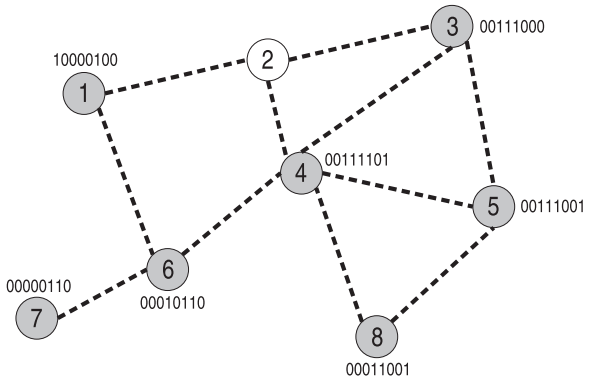


FIGURE 1
Occupied slots as seen by each node.

Note that this method solves the hidden station problem, because slot numbers are unique in a 2-hop neighborhood. This way, two nodes with a common neighbor will never transmit at the same time.

The number of slots can be chosen between 3 and 16 in our implementation, thus the size of `Occupied Slots` varies between 1 and 2 Byte. If more slots per frame are needed, the size of the field `Occupied Slots` grows. Thus far, the choice of slots is done in the same way as in LMAC, except for the fact that the synchronization started by a node rather than by the gateway.

### 2.2.2 Slot changes

The second difference is the fact that MLMAC stays adaptive even after slots are chosen. The last two fields of a control message are needed, because every node can start a synchronization. Due to this fact, it is possible that two distant nodes start a synchronization separately, as both of of them assume that they are the first to send. Their neighbors would synchronize with them and increase the `Age of Synchronization` by one before retransmitting. In this case, two different synchronizations would be flooding the net and meet somewhere in between the two starter nodes at some time in the future. At this point, nodes would realize that some of their neighbors use a different synchronization by comparing the field `Identity of Synchronization`. Now the field `Age of Synchronization` is compared. If the received value is equal to or higher than the local value saved in a node, this node becomes unsynchronized again and tries to find a new slot following the synchronization it just received. In this way, an older synchronization, which means one that has spread over more hops and thus a larger number of nodes, removes a younger one step by step. Once this procedure has begun, there will be a new set of nodes which follow the older synchronization every Frame, leading to a new set of nodes giving up their slot and so on until finally the whole network is synchronized.

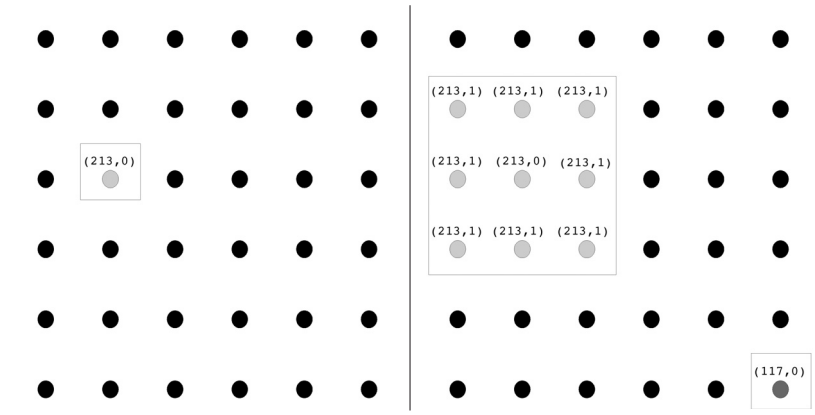


FIGURE 2
Starting of different synchronizations.

Figure 2 shows an example sensor network consisting of 6 times 6 nodes. On the left side of the figure it can be seen that node 213 wants to transmit a message, but does not have a synchronization yet. Therefore, it starts a new synchronization. The synchronization carries the identification of the starter node (213) and is transmitted to all neighboring nodes.

On the right side of the figure the retransmission of the synchronization from node 213 reaches all neighboring nodes, which increment the age of the synchronization by one before retransmitting. At the same time node 117 wants to transmit a message. As it is not synchronized yet (still in the `Wait`-state, see Section 2.4), it starts a synchronization of its own.

Figure 3 shows how the two different synchronizations spread through the network. On the left side of the figure the synchronization 213 has already

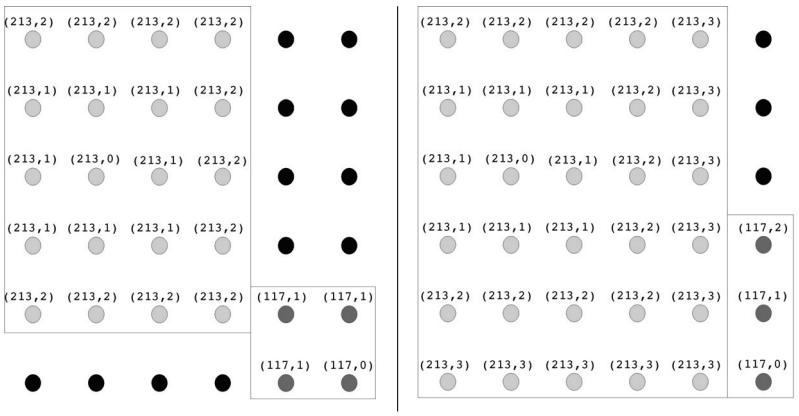


FIGURE 3
Two different synchronizations flooding the sensor network.

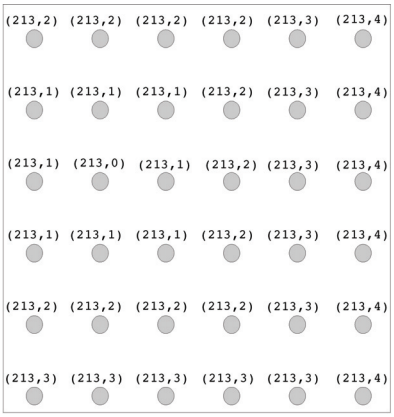


FIGURE 4
Two different synchronizations flooding the net.

spread to 20 nodes and has an age of 2 on the edge. The other synchronization, started by node 117, has only reached 4 nodes so far, with an age of 1.

When the two synchronizations collide at the nodes in the bottom right, the nodes compare the different ages. As the synchronization from node 213 is older, the nodes discard their existing synchronization and pick a new slot according to synchronization 213.

This can be seen on the right side of the figure. All nodes within 3 hops of node 213 are following its synchronization, only those not yet reached are unsynchronized or follow node 117. The last stage of the synchronization process can be seen on Figure 4, where all nodes follow the synchronization with the id 213. Now the whole network is synchronized and no more collisions occur, not even collisions of control messages. As long as no change in topology occurs, this is a stable state.

Due to the mobility of nodes, a node X may leave the radio range of node Y. Both nodes then realize that they do not receive any more control messages from each other and remove the other one from the field `Occupied Slots` which is transmitted in their control messages. When X moves into the radio range of another node Z, and Z knows a different node W that uses the same slot as X, the control messages of X and W collide at Z. Therefore, Z does not receive any more control messages in that slot and marks it as unused. Nodes X and W receive the control message from Z and realize that there must have been a collision of control messages. After this, they give up their current slot and try to find a different one.

### 2.3 Handling of unidirectional links

To determine whether a link is unidirectional or bidirectional, a neighbor list is used. In this list a counter is stored for every neighbor. When a node X receives a control message from node Y which does not contain the slot of

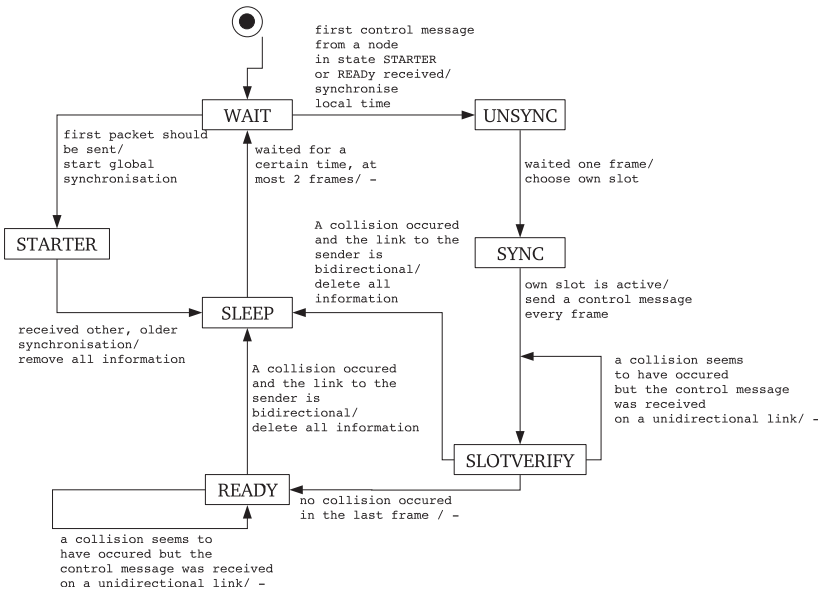


FIGURE 5
The finite state machine used in MLMAC.

node X, X increments the counter for Y in its neighbor list. If the received control message contains X's slot, the counter is decreased. The range of the counter is 0–N where N can be configured freely. Then, a threshold can be set, from where on the link will be counted as (partially) unidirectional. The need for this will become apparent when the state machine of MLMAC is discussed in Section 2.4

Ignoring or removing unidirectional links is a way often chosen by developers of MAC and routing protocols. We think that there are other ways which may be beneficial, and we are working on exploring some of them as outlined in our future work in Section 6.

### 2.4 The finite state machine of MLMAC

MLMACs state machine is shown in Figure 5. The rectangles represent states, the arrows transitions between them. The text on the arrows describes the necessary event for that transition and the action that is taken separated by a slash.

Initially, all nodes begin in the `Wait`-state. When they want to send a message without having received any control message previously, they change into the `Starter`-state. When only one node switches to starter, this is a stable state and the node remains there. If another node switched to the `Starter`-state earlier, this node gains knowledge of that fact after some time and switches to the `Sleep`-state from which it will return into the `Wait`-state after a certain time.

If a node receives a control message from another node in `Starter`- or `Ready`-state while it is in the `Wait`-state it synchronizes its local time with that of the originator of the control message and switches into the `Unsync`-state. After waiting one frame to overhear all transmitted control messages and calculate used slots, it chooses its own one and transitions into the `Sync`-state. When the chosen slot occurs the next time, the node starts to transmit its own control message. This is then repeated every frame and the node changes to state `Slotverify`. This state is used to verify that no other node has chosen the same slot during the last frame, which may have happened if multiple nodes were waiting for a new slot and the frame was relatively full.

As said before, the hardware used in the experiments does not enable us to detect collisions directly. Therefore, an indirect way of detection had to be implemented: A node X that has transmitted a control message can determine if a collision occurred by listening to its neighbors' control messages. If no collision occurred, the neighboring nodes have added X's slot to the field `Occupied Slots` in their control messages. Otherwise they did not. When X receives control messages containing its slot, it knows that no collision occurred because no other node has chosen the same slot. Therefore, it switches into the `Ready`-state.

Like the `Starter`-state, this is a stable state as long as no collision occurs. If a collision occurs, there must have been a mistake in the process of choosing slots, and the node returns into the `Sleep`- and finally into the `Wait`-state to start over again.

Note that MLMAC also distinguishes between collisions on unidirectional and bidirectional links. If a collision on a unidirectional link occurred on a node in `Slotverify`- or `Ready`-state, this node stays in the same state.

## 3 IMPLEMENTATION

### 3.1 Hardware

We used modified Lego RCX robots [5] to evaluate our protocol experimentally.

These robots feature a Renesas H8/300 micro-controller, which is a 16 Bit processor with a clock frequency of 16 MHz. The RCXs have three input channels and three output channels on top and an infrared module in the front. The Robots have been additionally equipped with a radio module of type ER400TRS [6] which we use instead of the included infrared module (IR) to enable broadcast protocols (see Figure 6). The ER400TRS has a range of up to 250 meters, operates in the Pan-European 433 to 434 MHz frequency band and uses a 3.6 V power supply. The over the air data rate is 19200 Baud. To enable the usage of the ER400TRS, an external connector to the serial port has been attached which allows us to connect either the IR or the radio module.

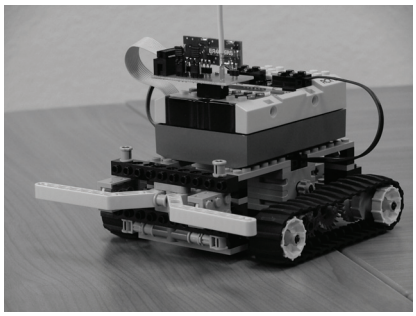

FIGURE 6
A modified RCX robot.

For our indoor experiments we reduced the range of the ER400TRS to about one meter. It may seem that the choice of radio module was not the best for our scenario, but when sensor nodes are designed, the chosen hardware is not always the best, either. Rather, it is normally the cheapest hardware that is chosen.

### 3.2 Software

Instead of the original software supplied with the Lego robots, we used REFLEX [7,8], an event driven operating system for deeply embedded systems in our experiments. We decided to use this operating system for 3 reasons. First, the resource constraints of wireless sensor networks are not much different from those of deeply embedded systems. Second, reflex was already ported for the H8/300 processor family when we started our work on the MLMAC. Third, it is implemented in C++, which leads to a number of advantages like, e.g., the usage of object oriented programming. We were able to concentrate on the features of MLMAC, without having to worry about the underlying platform.

On top of Reflex we used COPRA [9–11], a configurable communication framework. Its layered structure supplies an easy way of switching between different implementations for a single layer, which we used in our tests to switch between MLMAC and the simple probabilistic collision avoidance MAC we used for comparison.

## 4 EXPERIMENTS

We decided to experiment with a real sensor network right from the start for two reasons. First, to the best of our knowledge, LMAC has only been simulated. Second, we have observed drastic changes in link quality even in stationary sensor networks, as the authors of [12] have. Knowledge of these changes is vital for a MAC protocol, but we have not seen a realistic simulation model

thereof. Thus, we think it is necessary to experiment with real sensor nodes. For the same reason we did not simulate MLMAC but went to the real nodes right away.

In our experiments, we compared MLMAC to a competition based protocol, a simple collision avoidance protocol. As MLMAC is a TDMA based protocol, it should avoid collisions completely after a start period. A simple collision avoidance protocol on the other hand allows collisions but should produce a better throughput in low load situations, as nodes do not have to wait for their turn to send. Thus, latency should be smaller.

In CSMA/CA the channel is sensed before transmission, and if it is occupied the transmission is delayed for a random time. This reduces the probability of collisions. Due to the transceivers we used, the implemented MAC could not be a real CSMA/CA. The radio modules use an inbuilt checksum and only packets that were received flawlessly are handed to the software. As the radio modules only deliver whole packets, it is not possible to really listen to the medium. Therefore the channel is always assumed to be busy, and a random time is waited before the transmission of a packet. In this paper, the resulting protocol is called CA-MAC.

### 4.1 Static single-hop experiments

In the static single-hop experiments all robots were within direct radio range. The two MAC protocols were evaluated using 3, 5, and 9 robots. Every node wanted to transmit 50 packets with a size of 49 bytes to all other nodes. The size of the packet was chosen to represent some aggregated data received from attached sensors plus some header information. Node 1 started the transmission and every other node followed suit as soon as it received an initial packet. The number of slots was configured in MLMAC to match the number of nodes in the experiments. The size of a slot is statically determined by the size of the transmitted packets (data and control) and properties of the used radio modules (preamble size and data rate). This resulted in a slot length of 0.17 seconds and frame lengths of 1.53 seconds (for 9 nodes), 0.85 seconds (5 nodes) and 0.51 seconds (3 nodes). The CA-MAC sent at random intervals, but over the whole length of the experiment it transmitted the same number of packets as were sent in the MLMAC experiments.

Figure 7 shows the results of the static experiments. The experiments with 3 and 5 nodes were repeated three times, the ones with 9 nodes two times. The numbers shown on the figure are the averaged results. As you can see MLMAC always received all packets after the initialization phase was over. When using only three nodes, MLMAC even received all packets during initialization. Only during initialization with 5 or 9 nodes a few packets were dropped. The CA-MAC does not need to initialize, therefore no distinction is made. Due to the nature of the collision avoidance, the collision rate was high, though. Between a third and half of the packets were lost. This confirms that MLMAC is much better suited for our scenario than the CA-MAC.

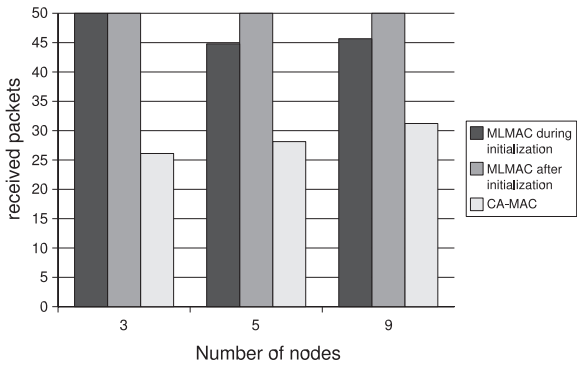


FIGURE 7
Average of received packets in the static experiment.

### 4.2 Static multi-hop line experiments

In preparation for the mobile experiments we continued with two different experiments using a line topology. For these experiments all 9 robots were arranged in a row with nodes 1 and 9 at the ends. To find proper distances between nodes, one node transmitted packets periodically and the next node was moved until it was unable to receive those packets. Then, it was placed a step nearer to the transmitting node. The experiments were performed outdoors, on a concrete road. In the first experiments, node 1 transmitted 50 packets and node 9 counted how many arrived. Each node in between counted how many were forwarded and how many were dropped. The packets were sent in time intervals equaling the length of one, three and five slots of MLMAC in three different experiments. In the second line experiment node 1 started transmitting 50 packets and once the first packet reached node 9, it started to transmit 50 packets, too. For the MLMAC, this was only done with sending intervals of 3 or 5 slots, as it is impossible to transmit 2 packets every slot without using some further means like, e.g., aggregation. The packets were forwarded using a simple duplicate suppression algorithm. When a node received a message it remembered the originator (node 1 or 9) and the sequence number of that packet. When a packet with the same or a lower sequence number was received later, it was discarded, if its number was higher it was retransmitted. All experiments were conducted with MLMAC and with CA-MAC and repeated 9 times for each of them. Again, the CA-MAC sent at random intervals, but over the whole length of the experiment these intervals evened out to those used in the MLMAC experiments.

Figure 8 shows some results from the first experiments. If each node would only be able to receive messages from its direct neighbors, all graphs should have been steadily decreasing or stayed at the same level as a packet that was lost at, e.g., node 3 could never be received at the following nodes 4 to 9. Instead the figure shows that, e.g., when using the CA-MAC with a

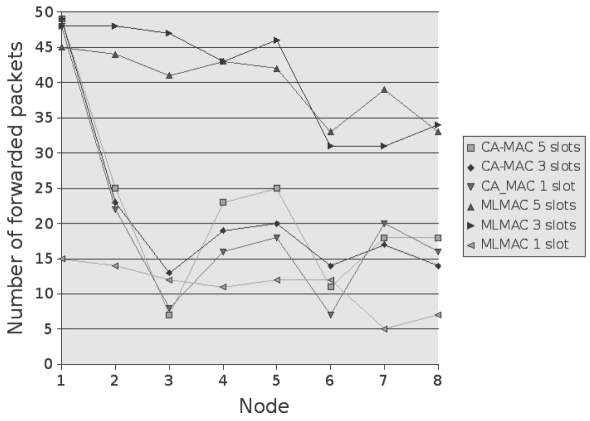


FIGURE 8
Forwarded packets for each node. The packets originate at node 1 and stop at node 9.

transmission interval of roughly 5 slots node 3 received only 8 messages while its neighbors received 19 (node 2) and 28 (node 4). This reveals a strong variation in radio range of the sending nodes. In a few cases, node 9 even received packets from node 1 directly. The overall low throughput can be explained by the same fact because the number of slots per frame was set to low for the MLMAC, as we assumed that 3 or at most 4 slots would be sufficient. For the CA-MAC the variation in radio range produced more collisions as more nodes were able to disturb each other. Still, even with a much too small number of slots, the results of MLMAC are much better than those of the CA-MAC when messages are transmitted every 3 or 5 slots.

Figure 9 shows the number of packets originating at node 1 forwarded by each node in the second experiments, while Figure 10 shows the same

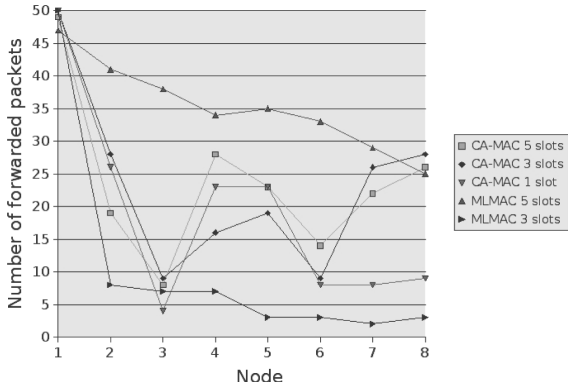


FIGURE 9
Forwarded packets for each node. The packets originate at node 1 and are forwarded to node 9. Node 9 sends to node 1 at the same time.

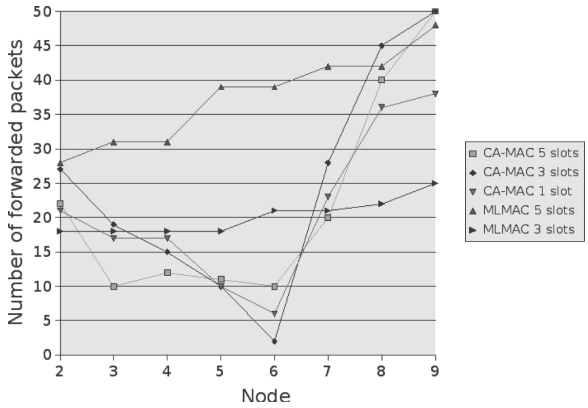


FIGURE 10
Forwarded packets for each node. Packets originate at node 9 and are forwarded to node 1. Node 1 sends to node 9 at the same time.

for those originating at node 9. The results are similar to each other and to those of the first line experiments. A big difference can be seen in the results using MLMAC when sending every 3 slots. Obviously, node 1 had problems finding its slot as 42 messages were lost from node 1 to node 2. Otherwise, they confirm the results obtained earlier. The radio range is not stable and therefore the number of slots was too small.

### 4.3 Static multi-hop square experiments

After we saw the variations in link quality and radio range in the line experiment, we performed some experiments to determine the stability of links in time. Therefore, we arranged nine robots in a square of three times three with a distance of ten meters between neighbors. This distance was determined as above. After placement, we tried to build a routing tree and send messages from various starter nodes. As we assumed that the node in the middle would be heard by all, we only started building the trees from the nodes on the corners. The nodes one, three, four and seven which were on the corners of the square repeatedly tried to build a routing tree. Once this tree was established 10 messages were sent. After this, the number of received messages on each node was determined.

On Figure 11 you see two routing trees, which were obtained in our experiments. On the left side you can see a routing tree that nearly follows the theory. Direct neighbors of node 1 can communicate directly, the nodes 8 and 4 that are farthest away from 1 need a second hop via node 5. Node 6 experienced some problems and was not able to transmit to node 1 even though it could hear node 1 directly. This is shown as a dashed lines which represents an unidirectional link which would normally have been discarded. On the right side you see a different routing tree obtained from the same starting node a little

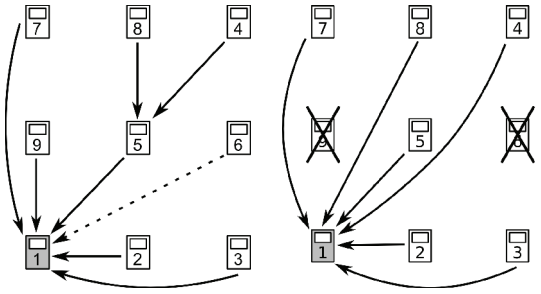


FIGURE 11
Two RCX routing trees started from node 1 at different points in time.

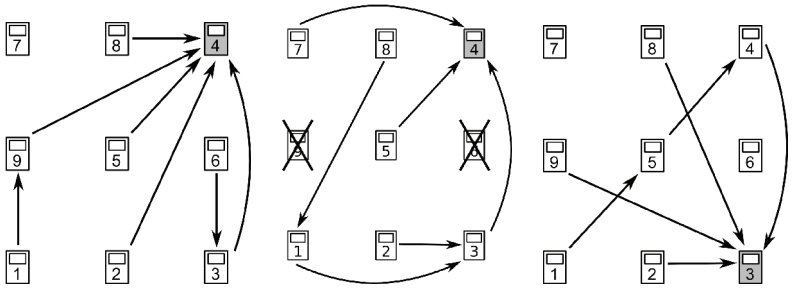


FIGURE 12
Two routing trees started from node 4 at different points in time and one started from node 3.

time later. Now, all nodes can communicate directly with node 1 except for nodes 6 and 9 which have no connectivity at all. While these two routing trees are not optimal, they somehow still represent the expected layout.

On Figure 12 you see the same experiment again, this time started from node 4. The left side of Figure 12 is very much the same as the right side of Figure 11. All nodes communicate directly with node 4, except for node 7 which does not communicate at all, node 1 which is connected through node 9 and 6 through 3. The middle of the figure shows a completely different picture. There are only three nodes which communicate directly with node 4, the nodes 3, 5 and 7. Node 2 is connected via node 3 which is acceptable, even that node 1 uses node 3 to reach node 4 is no problem. The problem is, that node 8, which should be a direct neighbor of node 4 needs three hops to connect. This was no single effect either – when we started the protocol from node 3 which can be seen on the right side, node 1 needed three hops to connect, too. The trees obtained from node 7 were not much better, either. Some routing trees we obtained did not even reach half the nodes.

As said before, after building the trees we sent 10 messages from each source and counted the number of arriving messages. For these experiments, only the best two routing trees from each node were used. Note that the ones

| | | |
|---|---|---|
| 10 | 10 | 10 |
| 10 | 10 | 0 |
| ***10*** | 10 | 10 |

| | | |
|---|---|---|
| 10 | 3 | 1 |
| 0 | 10 | 0 |
| ***10*** | 10 | 10 |

FIGURE 13
Number of received messages out of 10 using the routing trees in Figure 11.

| | | |
|---|---|---|
| 0 | 10 | ***10*** |
| 3 | 10 | 10 |
| 2 | 10 | 10 |

| | | |
|---|---|---|
| 8 | 3 | ***10*** |
| 0 | 8 | 0 |
| 8 | 8 | 8 |

| | | |
|---|---|---|
| 0 | 8 | 10 |
| 8 | 9 | 10 |
| 8 | 10 | ***10*** |

FIGURE 14
Number of received messages out of 10 using the routing trees in Figure 12.

discussed before are those best trees, some others were quite disastrous. The number of received messages varied between 1 and 10 out of 10. In the case of the routing tree shown in Figure 11 on the right side node 4 received only one message from node 1. All results for both routing trees can be seen in Figure 13, the italic numbers represent the starter nodes. In the case of the routing tree shown in Figure 12 on the left side, node 8 received three messages from node 4. This seems strange as the range of the longest hop is 22, 4 meters instead of 28.3 meters but the number of hops increased from one to three. When node 3 was master and node 1 had a distance of three hops with a longest hop length of 20 meters eight packets were received out of ten. The total number of received packets for all three routing trees is shown in Figure 14.

Similar effects have been observed in [12]. The authors describe an outdoor experiment in which they used 24 ScatterWeb ESBs [13] in an area of 80 times 140 meters. Every hour, a routing tree was built, and information sent to a sink. The experiment ran for three weeks. Often neighboring nodes did not hear each other while distant nodes did. In some cases these did not even have line of sight but were blocked by a building.

We have not yet seen any simulator that could produce such behavior, and we have not found any model to describe these effects either. All these results confirmed that we needed to evaluate MLMAC on a real sensor net in the mobile scenario.

### 4.4 Mobile experiments

In the mobile experiment the RCX robots moved around a room of 6 times 8 meters containing normal bureau furnishing with a speed of 6 meters per minute in a nearly random manner. Initially, they only moved forward but every time they hit an obstacle, they moved back a pace, turned for a random time in a random direction and started moving again. As all nodes started with a different facing, the mobility soon led to a multi-hop environment. The application was the same as in the static experiment. 50 Packets with a size of 49 Bytes were transmitted by each node and node 1 started the experiment. Due to mechanical failure, we could use only 8 robots, not 9 as in the

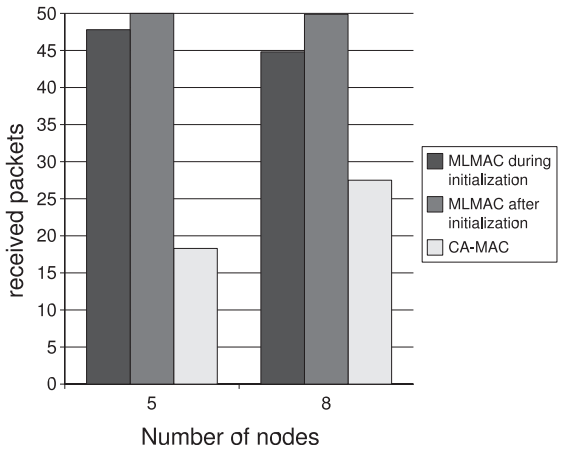


FIGURE 15
Average of received packets in the mobile experiment.

static experiment. This resulted in a frame length of 1.36 seconds. Also, the Experiment with 3 nodes was ignored, as there would be no gain from it.

Figure 15 shows the results of the mobile experiments. Again, after the initialization, MLMAC delivered all packets flawlessly. During the initialization phase a few packets have been lost as was expected. The CA-MAC delivered between 18 and 27 percent of the packets. When the results are compared to the static experiment, it can be seen that mobility can be a challenge for medium access, the CA-MAC suffers heavily. MLMAC on the other hand performed nearly the same when mobility was used.

### 4.5 Program code size

Table 3 shows the size of the whole program code of a simple application that uses MLMAC or CA-MAC on the RCX robots. Note that this size depends on the architecture in use as the size of primitive data types and pointers in C++ varies on different platforms. The important part that you can see in the table is that MLMAC needs only about 4 kilobyte more than CA-MAC. This is less than a third of the size of CA-MAC. The overhead in program code size is acceptable for the gain in performance as shown earlier. Note also that the

| MAC protocol | text | data | bss | total |
|---|---|---|---|---|
| MLMAC | 14072 | 456 | 3522 | 18050 |
| CA-MAC | 10116 | 402 | 3376 | 13888 |

TABLE 3
Program code sizes of a simple application using MLMAC or CA-MAC, including operating system and drivers

size measured is for our feasibility study implementation and we are positive that it could be reduced by careful re-implementation.

## 5 RELATED WORK

There have been numerous proposals of MAC layers for wireless sensor networks. These include contention based ones as well as plan based ones. Prominent examples of contention based protocols are S-MAC [14], T-Mac [15] and D-Mac [16]. Examples of plan based ones are LMAC [4], AI-LMAC [17], LooseMAC and TightMAC [18].

S-MAC or Sensor-MAC [14] is a variation of the DCF mode used in the IEEE 802.11 standard, which has been optimized for low energy consumption. It uses the same type of packets (RTS, CTS, ACK and DATA), but their meaning is changed a little. Instead of sending an RTS for each data packet, the medium is reserved (RTS) and confirmed (CTS) for a whole stream of them in S-MAC. While this already leads to a reduction of packets, the main energy consumption is achieved by introducing sleep states. All nodes change from sleep to active regularly, and neighboring nodes synchronize their sleeping times as much as possible to ensure fast distribution of messages. Otherwise, an active node would have to extend its waking period until the neighboring node it wants to transmit to wakes up.

T-MAC [15] is an extension of S-MAC. In S-MAC, the sleep- and active period have a fixed size. Nodes that are in an active state remain active, even if no messages need to be transmitted to or from them. This is where the optimizations of T-MAC take hold. Instead of a fixed active time, a flexible one is used. In the beginning of its active state, a node transmits or listens to the medium. When it does not need to transmit any more packets and no other node wants to transmit to this node, it goes to sleep. This mechanism concentrates all transmissions at the beginning of the active phases of the nodes, which leads to a longer sleep period, but also to more collisions of RTS packets.

The main focus of D-MAC [16] is the efficient forwarding of data packets. S-MAC and T-MAC often introduce a sleep delay, because the next node may still be asleep when another wants to transmit data. This is especially dangerous, if all nodes need to transmit to a single sink (tree topology), which leads to timing problems and packet loss due to buffer overflow. D-MAC uses a data tree model, where nodes on the path to the sink wake up in a chain reaction, removing the sleeping delay. This is possible, because every node only accepts data packets from its children and forwards them only to its parent. This allows dividing each cycle into three phases: In the fist, active phase a node receives messages. In the second phase it forwards the message. After receiving the acknowledge for the message, the node transitions into phase three and sleeps.

All of the protocols described above are not suitable for our scenario. S-MAC and T-MAC would suffer from the high load and low bandwidth.

D-MAC works only for a single sink, and needs fairly stable links. All three protocols have problems with the mobility of nodes we assume. Therefore, we choose to investigate a plan based protocol.

The authors of [4] have introduced LMAC, the lightweight medium access protocol for wireless sensor networks, which is the basis of this work. While some of their assumptions do not hold for our scenario, the main idea of representing the slots used by the neighboring nodes as a bitmask can be found in MLMAC, too. One of their features, turning radio modules off to save energy, is not implemented in the current version of MLMAC. It could be added without much afford, though, if different hardware was used.

AI-LMAC is introduced in [17]. It is an enhancement of LMAC which allows dynamic reallocation of slots, depending on the network load. The authors assume a routing tree which leads to a sink and optimize the slot usage along the branches of this tree. This is realized by the usage of so called Data Distribution Tables, which are used to determine the network load which results after a query from the sink. With this information, slots can be reserved according to the presumed needs. This approach does not fit into our scenario however, as we do not want to use a single sink. Rather, all nodes communicate among each other. Another difference is that MLMAC allows free choice of routing protocol.

Another plan based protocol is LooseMAC [18]. In this protocol, the node density is assumed to be known in advance. This enables a node to calculate the size of a frame and the number of slots in each frame. Once this is done, a newly started node switches to the `NEWSLOT`-state when it has chosen its slot. This is the first of three states a node can reach. A node in this state transmits a control message containing its ID and two status bits in its chosen slot. After transmitting this message, the node transitions into the `WATCH`-state. In this state it listens to the medium for one frame to determine if a collision occurred. If it did, the node starts again by selecting a new slot. Otherwise it reaches the `Ready`-state. Once all nodes are in this state, no more collisions can occur.

TightMAC [18] is an extension of LooseMAC. TightMAC uses the same states as LooseMAC, but adds the states `Ready-1`, `Ready-2` and `Ready-3`, which can be reached by a node once all its neighbors within a certain distance have reached the `Ready`-state. A node reaches the state `Ready-1` once all of its direct neighbors are in `Ready`-state. This incremental behavior is also true for the other two states. Once a node reaches the `Ready-3`-state, it knows that all its neighbors within two hops have at least the `Ready-1`-state. Now this node transmits the number of its direct neighbors. Once it has received this message from all of its neighbors, too, it can calculate the minimum number of slots needed in a frame. With this information, the already used frames from LooseMAC can be split into multiple TightMAC frames. This way, neighboring nodes can have frames of different sizes, which makes the protocol more complicated, but the throughput is increased. Seen from the point of a LooseMAC frame, one node may now occupy more than one slot.

The knowledge about neighbors in the two hop distance is important to find a good schedule, and is also used in MLMAC. Also, the state machines described in LooseMAC and TightMAC inspired us during the work on MLMAC.

## 6 CONCLUSION AND FUTURE WORK

In this paper we have introduced Mobile LMAC, a novel TDMA protocol for mobile wireless sensor nodes. Mobile LMAC is based on the LMAC protocol but in our scheme each node can spontaneously establish a TDMA schedule on demand or join/leave existing schedules while nodes are moving. Thus a high channel throughput can be achieved for mobile sensor nodes even in heavy load situations. We have shown the feasibility of our protocol by means of experiments with a real mobile sensor network based on modified RCX robots. Furthermore, we compared the results with similar experiments using a simple collision avoidance MAC. The latter is reasonably well suited for sporadic communication between mobile nodes and does not require the exchange of additional protocol data. However, in the case of high load the Mobile LMAC achieved a far better result.

In the future we will derive simulation models from our experiments and investigate the effectiveness of Mobile LMAC in large multi hop settings. We also plan to port MLMAC to other sensor nodes, which we can mount on top of the RCX robots like we did with the ScatterWeb Embedded Sensor Board (ESB) [13], (see Figure 16). This way, the RCX robots will only be the platform for mobility and the radio module of the sensor node can be used to experiment with power savings as in LMAC.

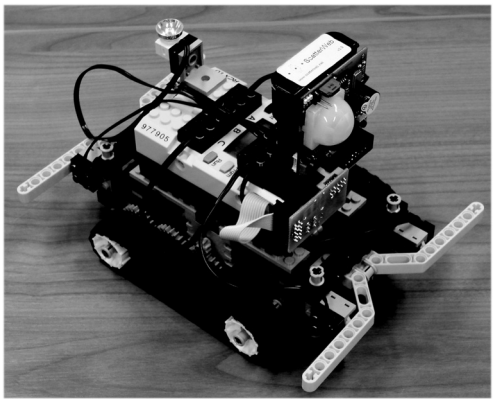

FIGURE 16
An RCX with an mounted ESB.

At the moment, a different matter is being addressed. In wireless sensor networks, the used cheap transmitters often lead to unreliable connections between nodes. Studies on real hardware have shown that unidirectional links are fairly common, sometimes temporary, sometimes permanent. It has also been shown that these links often bridge a far wider range than bidirectional links. Therefore, we are working on a way to exploit this longer range. Using the unidirectional links instead of discarding them results in a number of change necessary in the state machine of MLMAC and the need for a way to send acknowledges over multiple hops. Currently, a cooperation with a routing layer is investigated to solve this problem.

## REFERENCES

[1] Ioannis Chatzigiannakis, Sotiris Nikoletseas and Paul Spirakis. Smart dust protocols for local detection and propagation. In *POMC '02: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*. ACM Press, New York, NY, USA, pp. 9–16, 2002.

[2] Ioannis Chatzigiannakis, Sotiris Nikoletseas and Paul G. Spirakis. Efficient and robust protocols for local detection and propagation in smart dust networks. *Mob. Netw. Appl.*, **10**(1–2) (2005), 133–149.

[3] J. M. Kahn, R. H. Katz and K. S. J. Pister. Next century challenges: mobile networking for "smart dust". In *MobiCom '99: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*. ACM Press, New York, NY, USA, pp. 271–278, 1999.

[4] L. F. W. van Hoesel and P. J. M. Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *INSS, Japan*, June 2004.

[5] Holly Patterson-McNeill and Carol L. Binkerd. Resources for using lego mindstorms. In *Proceedings of the Seventh Annual Consortium for Computing in Small Colleges Central Plains Conference on The Journal of Computing in Small Colleges*, pp. 48–55, USA, 2001. Consortium for Computing Sciences in Colleges.

[6] circuit design inc. low power radio solutions. www.lprs.co.uk/main/product.info.php?productid = 154.

[7] Karsten Walther, Reinhard Hemmerling, and Jörg Nolte. Generic trigger variables and event flow wrappers in reflex. In *ECOOP – Workshop on Programming Languages and Operating Systems*, June 2004.

[8] Karsten Walther and Jörg Nolte. Event-flow and synchronization in single threaded systems. In *Proceedings of First GI/ITG Workshop on Non-Functional Properties of Embedded Systems (NFPES)*, March 2006.

[9] Reinhardt Karnapke and Jörg Nolte. Copra – a communication processing architecture for wireless sensor networks. In *Euro-Par 2006 Parallel Processing*. Springer, pp. 951–960, 2006.

[10] Marcin Brzozowski, Reinhardt Karnapke, and Jörg Nolte. Impact – a family of cross-layer transmission protocols for wireless sensor networks. In *The First International Workshop on Research Challenges in Next Generation Networks for First Responders and Critical Infrastructures (NetCri 07), in Conjunction with 26th IEEE IPCCC*, 2007.

[11] Stephan Mank, Reinhardt Karnapke, and Jörg Nolte. An adaptive tdma based mac protocol for mobile wireless sensor networks, best paper award. In *International Conference on Sensor Technologies and Applications*, 2007.

[12] Turau, Renner, and Venzke. The heathland experiment: Results and experiences. In *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks.*, June 2005.

[13] Jochen Schiller, Achim Liers, Hartmut Ritter, Rolf Winter, and Thiemo Voigt. Scatterweb – low power sensor nodes and energy aware routing. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, 2005.

[14] Wei Ye, John Heidemann, and Deborah Estrin. *An Energy-Efficient Mac Protocol for Wireless Sensor Networks*, pp. 1567–1576, 2002.

[15] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ACM Press, New York, NY, USA, pp. 171–180, 2003.

[16] Gang Lu, Bhaskar Krishnamachari, and Cauligi S. Raghavendra. An adaptive energy-efficient and low-latency mac for data gathering in sensor networks. In *Int. Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, 2004.

[17] P.J.M. Havinga S. Chatterjea, L.F.W. van Hoesel. Ai-lmac: An adaptive, information-centric and lightweight mac protocol for wireless sensor networks.

[18] Costas Busch, Malik Magdon-ismail, Fikret Sivrikaya, and Blent Yener. Contention-free mac protocols for wireless sensor networks. In *DISC*, pp. 245–259, 2004.