Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks

Ansgar Fehnker[1,⋆], Lodewijk van Hoesel[2], and Angelika Mader[2,⋆⋆]

[1] National ICT Australia and University of New South Wales, Australia
ansgar.fehnker@nicta.com.au
[2] Department of Computer Science, University of Twente, The Netherlands
l.f.w.vanhoesel@utwente.nl, mader@ewi.utwente.nl

Abstract. In this paper we report on modelling and verification of a medium access control protocol for wireless sensor networks, the LMAC protocol. Our approach is to systematically investigate all possible connected topologies consisting of four and of five nodes. The analysis is performed by timed automaton model checking using Uppaal. The property of main interest is detecting and resolving collision. Evaluation of this property for all connected topologies requires more than 8000 model checking runs. Increasing the number of nodes would not only lead increase the state space, but to a greater extent cause an instance explosion problem. Despite the small number of nodes this approach gave valuable insight in the protocol and the scenarios that lead to collisions not detected by the protocol, and it increased the confidence in the adequacy of the protocol.

1 Introduction

In this paper we report about modelling and verification of a medium access control protocol for wireless sensor networks, the LMAC protocol [10]. The LMAC protocol is designed to function in a multi-hop, energy-constrained wireless sensor network. It targets especially energy-efficiency, self-configuration and distributed operation. In order to avoid energy-wasting effects, like idle listening, hidden terminal problem or collision of packets, the communication is scheduled. Each node gets periodically a time interval (slot) in which it is allowed to control the wireless medium according its own requirements and needs. Here, we concentrate on the part of the protocol that is responsible for the distributed and localised strategy of choosing a time slot for nodes.

Although, the basic idea of the protocol is quite simple, the possible behaviours get quickly too complex to be overseen by pure insight. Therefore, we chose a model checking technique for the formal analysis of the protocol. We apply model checking in an experimental approach [4,6]: formal analysis can only increase the *confidence* in the correctness of an implementation, but not

⋆ National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.
⋆⋆ supported by NWO project 632.001.202, Methods for modelling embedded systems.

J. Davies and J. Gibbons (Eds.): IFM 2007, LNCS 4591, pp. 253–272, 2007.

guarantee it. This has two reasons: first, a formal correctness proof is only about a model, and not about the implementation. Second, we will (and can) not prove correctness for the general case, but only for instances of topologies.

Model checking as a way to increase the confidence comes also into play, as we do not aim to prove that the protocol is correct for all considered topologies. This is in contrast to related work on verification of communication protocols, such as [1]. It is known beforehand that there exist problematic topologies for which the LMAC protocol cannot satisfy all relevant properties. The aim is to iteratively improve the model, and to reduce the number of topologies for which the protocol may fail. This is an important quantitative aspect of the model checking experiments presented in this paper.

In order to get meaningful results from model checking we follow two lines:

Model checking experiments: We systematically investigate all possible connected topologies of 4 and 5 nodes, which are in total 11, and 61 respectively. For 12 different models and 6 properties we performed about 8000 model checking runs using the model checker Uppaal [2,3]. There are the following reasons for the choice of the model checking approach considering all topologies:

(1) Relevant faults appear already in networks with a small number of nodes. Of course, possible faults that involve more nodes are not detected here.

(2) It is not enough to investigate only representative topologies, because it is difficult to decide what "representative" is. It turned out that topologies that look very "similar" behave differently, in the sense that in one collision can occur, which does not in the other. This suggests that the systematic way to investigate all topologies gives more reliable results. This forms a contrast to similar approaches such as [8] which considers only representative topologies, and the work in [5], which considers only very regular topologies.

(3) By model checking all possible scenarios are traversed exhaustively. It turned out that scenarios leading to collisions are complex, and are unlikely to be found by a simulator. On the other hand, simulations can deal with much higher numbers of nodes. We believe that both, verification and simulation, can increase the confidence in a protocol, but in complementary ways.

Systematic model construction: The quality of results gained from model checking cannot be higher than the quality of models that is used. We constructed the models systematically, which is presented in sufficient detail. We regard it as relevant that the decisions that went into the model construction are explicit, such that they can be questioned and discussed. It also makes it easier to interpret the result of the model checking experiments, i.e. to identify what was proven, and what not. The reader who is not interested in the details of the model should skip therefore Section 4.

The goal of the protocol is to find a mapping of time slots to nodes that prevents collisions. To this end it is necessary that not only direct neighbours have different slots, but also that all neighbours of a node have pairwise different slots. Neighbours of neighbours will be called *second-order* neighbours. The problem

is at least NP-hard [7,9]: each solution to the slot-mapping problem is also a solution to the graph colouring problem, but not vice versa.

When starting the protocol analysis using Uppaal, the protocol had been developed [10], implemented and analysed by simulation. The specification consisted of the publication mentioned, and personal explanations. Our analysis here restricts to the fragment of the protocol concerned with the slot distribution mechanism trying to avoid collision. Other aspects, as time synchronisation or sleeping modes of nodes, are covered by the protocol, but are not addressed in the analysis here. During our modelling and verification efforts we found that the implementation covered more aspects than the specification did. The main results of our analysis were an improvement of the protocol, such that less collisions remain undetected, and an analysis of possible undetected collisions showing that undetected collisions do not prevent connection to the gateway.

The paper is structured as follows. In Section 2 we give a short description of the LMAC protocol, and in Section 3 a brief introduction to timed automata. The models and properties are described in detail in Section 4. The model checking results are discussed in Section 5. We conclude with discussions in Section 6.

2 The LMAC Protocol

In schedule-based MAC protocols, time is organised in *time slots*, which are grouped into *frames*. Each frame has a fixed length of a (integer) number of time slots. The number of time slots in a frame should be adapted to the expected network node density or system requirements.

The scheduling principle in the LMAC protocol [10] is very simple: every node gets to control one time slot in every frame to carry out its transmission. When a node has some data to transmit, it waits until its time slot comes up, and transmits the packet without causing collision or interference with other transmissions. In the LMAC protocols, nodes always transmit a short control message in their time slot, which is used to maintain synchronisation.

The control message of the LMAC protocol plays an important role in obtaining a local view of the network within a two-hop distance. With each transmission a node broadcasts a bit vector of slots occupied by its (first-order) neighbours . When a node receives a message from a neighbour, it marks the respective time slots as occupied. To maintain synchronisation other nodes always listen at the beginning of time slots to the control messages of other nodes.

In the remainder we will briefly describe the part of LMAC concerned with the choice of a time slot. We define four operational phases (Fig. 1):

Initialisation phase (I) — The node samples the wireless medium to detect other nodes. When a neighbouring node is detected, the node synchronizes (i.e. the node knows the current slot number), and proceeds to the wait phase W, or directly to the discover phase D.

Wait phase (W) — We observed that ,especially at network setup, many nodes receive an impulse to synchronize at the same time. The protocol introduces randomness in reaction time between synchronising with the network and

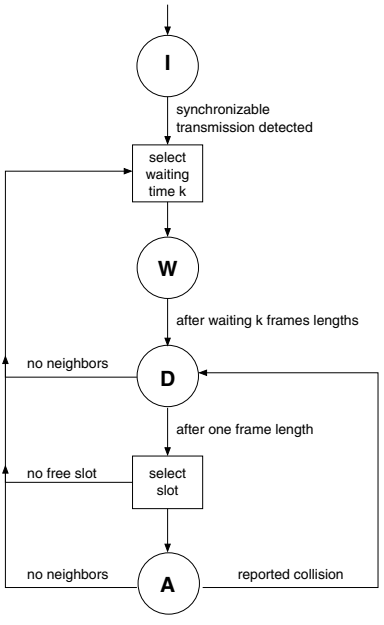

Fig. 1. Control flow diagram of the protocol

actually choosing a free time slot, to reduce the likelihood that nodes select slots at the same time. This is achieved by inserting a random wait time after the initialisation phase I and before the discover phase D.

Discover phase (D) — The node collects first-order neighbourhood information during one entire frame and records the occupied time slots. If all information is collected, the node chooses a time slot and advances to the active phase A.

By performing an 'OR'-operation between all received bit vectors, a node in the discover phase D can determine which time slots in its second-order neighbourhood are unoccupied and can be freely used. At this moment the node can choose *any* time slot that it marked as unoccupied. To reduce the probability of collisions, the protocol is to randomly choose one of the available slots.

Active phase (A) — The node transmits a message in its own time slot. It listens in all other time slots and accepts data from neighbouring nodes. The node also keeps its view on the network up-to-date. When a neighbouring node informs that there was a collision in the time slot of the node, it will return to the wait phase W. Collisions can occur when two or more nodes choose the same time slot for transmission simultaneously. This can happen with small probability at network setup or when network topology changes due to mobility of nodes.

The nodes that cause a collision cannot detect the collision by themselves; they need to be informed by their neighbouring nodes. These neighbouring nodes use their own time slot to inform the network that they detected a collision. When a node is informed that it is in a collision it will give up its time slot and return to the discover phase D.

3 Timed Automata

Systems are modelled in Uppaal as a parallel composition of timed automata [3]. Time is modelled using real-valued clocks and time only progresses in the locations of the automata: transitions are instantaneous. The guards on transitions between locations in the automata and the invariants in the various locations may contain both integer-valued variables and real-valued clocks. Clocks can be reset to zero on transitions. Several automata can synchronize on transitions using handshake and broadcast synchronisation. Shared variables can be used to model data transfer between automata. Locations can be declared urgent, which means time is not allowed to progress, or committed, which means time is not allowed to progress and interleaving is restricted. If only one automaton is in a committed location at any one time, its transitions are guaranteed to be atomic.

Properties of systems are checked by the Uppaal model checker, which performs an exhaustive search through the state space of the system for the validity of these properties. It can check for invariant, reachability, and liveness properties of the system, specified in a fragment of TCTL.

4 Models and Properties

4.1 Model Decomposition

Uppaal models are, as mentioned in the previous section, parallel compositions of timed automata, and allow for compositional modeling of complex systems. The LMAC protocol is naturally distributed over the different nodes. The Uppaal model reflects this by including exactly one timed automaton model for each node. Each of these timed automata models is then organised along the lines of the flow chart in Section 2.

The Uppaal model of the LMAC protocol will be used to analyse the behaviour, correctness and performance of the protocol. Since the LMAC protocol builds on an assumed time synchronisation, the Uppaal model will also assume an existing synchronisation on time. Although it would be interesting to analyse the timing model in detail, it falls outside of the scope of the protocol and this investigation.

The LMAC protocols divides time into frames, which are subdivided into slots. Within a slot, each node communicates with its neighbours and updates its local state accordingly. We model each slot to take two time units. Each node has a local clock. Nodes communicate when their local clock equals 1, and update information when their clocks equals 2. At this time the clock will be reset to zero.

Based on this timing model, the protocol running on one node is modelled as a single timed automaton. The complete model contains one of these automata for each node in the network. The timed automata distinguish between 4 phases, as shown in the control flow graph in Figure 1. The first phase is the initialisation phase, the second the optional wait phase. The next part models the discover

phase which gathers neighbourhood information. At the end of the discover phase a node chooses a slot, and proceeds to the fourth and last phase, the active phase. Figure 2 to 6 depict the models for each phase. Details of the different parts will be discussed later in this section. Note, that the model presented here serves as a baseline for an iterative improvement of model and protocol.

Channels and Variables

Global channels and variables. The wireless medium and the topology of the network are modelled by a broadcast channel `sendWM`, and a connectivity matrix `can_hear`. A sending node `i` synchronises on transitions labeled `sendWM!`. The receiving nodes `j` then synchronizes on label `sendWM?` if `can_hear[j][i]` is true. This model of *sending* is used in the active phase (Fig. 6), and the model of *receiving* during initialisation (Fig. 2), discover (Fig. 4) and active phase (Fig. 6).

The model uses three global arrays to maintain a list of slot numbers and neighbourhood information for each node. Array `slot_no` records for each node the current slot number. Array `first` and `second` record for each node information on the first and second-order neighbours, respectively. Note, that the entries of these arrays are bit vectors, and will be manipulated using bit-wise operations. All nodes have read access to each of the elements in the arrays, but only write access to its own. The arrays are declared globally to ease read access.

The model uses two additional global variables `aux_id` and `aux_col`. These are one place buffers, used during communication to exchange information on IDs and collisions.

Local variables. Each node has five local variables. Variable `rec_vec` is a local copy of received neighbourhood information, `counter` counts the number of slots a node has been waiting, and `current` the current slot number, with respect to the beginning to the frame. Variable `col` records the reported collisions, while `detected` is used to record detected collisions. Finally, each node has a local clock `t`.

The node model. The remainder of this section will discuss each part of the node model in detail.

Initialisation phase. The model for the initialisation phase is depicted in Figure 2. As long a node does not receive any message it remains in the initial node. If a node receives a message, i.e. if it can hear (`can_hear[id][aux_id]==1`) and synchronise with the sender (`sendWM?`), it sets its current slot number to the slot number of the sender (`current=slot_no[aux_id]`), and resets its local clock (`t=0`). The slot number of the sender is part of the message that is send. From this time on the receiver will update the current slot number at the same rate as the sender. They are equal whenever either of them sends. This synchronisation is the subject of one of the properties that will be verified later.

If the receiver receives a second packet before the end of the slot a collision has occurred. The node will discard the received information and return to the initial

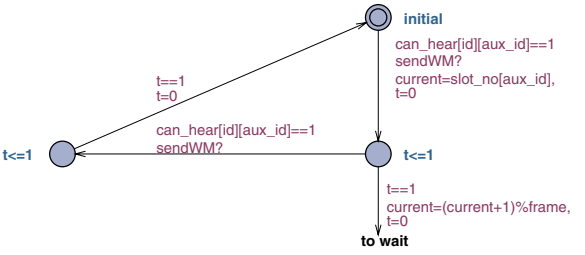

Fig. 2. Model of the initialisation phase

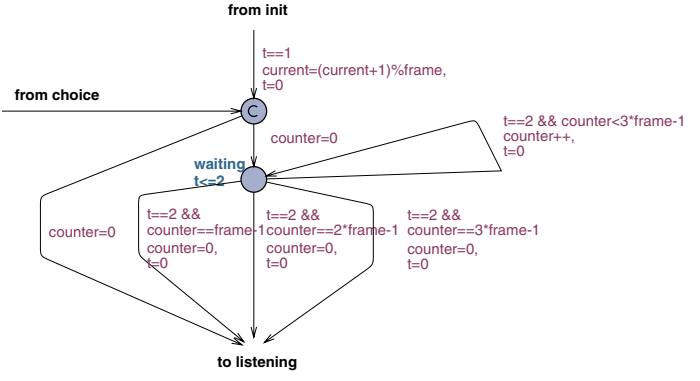

Fig. 3. Model of the wait phase

location. If no collision occurs, the node will proceed to the next slot, increment the current slot counter modulo the length of the frame (`current=current+1%frame`), and proceed to the wait phase (Figure 3).

Wait Phase. When a node enters the wait phase, it may decide (non-deterministically) to skip this phase. A node waits for at most 3 frames in this location `waiting`. Waiting is implemented as a self loop, which is guarded by `counter<3*frame-1`. The loop increments the counter at the end of a slot (`t==2`). A node can proceed to the discover phase when it waited for exactly one, two or three frames.

Discover Phase. The model for the discover phase consists of four locations (Figure 4). The entry location `listening0` models when a node is sensing the medium. Location `rec_one0` models that a node continues sensing after reception of a first message. Location `done0` is reached when a node detected a collision. Finally, the model contains a committed location, in which the node checks if it listened to the medium for a full frame. If it did, it proceeds to choose a free slot, otherwise it continues listening.

Clocks and variables will be updated as follows. When a node enters location `listening0`, the local clock will be zero. It will wait in this location for at

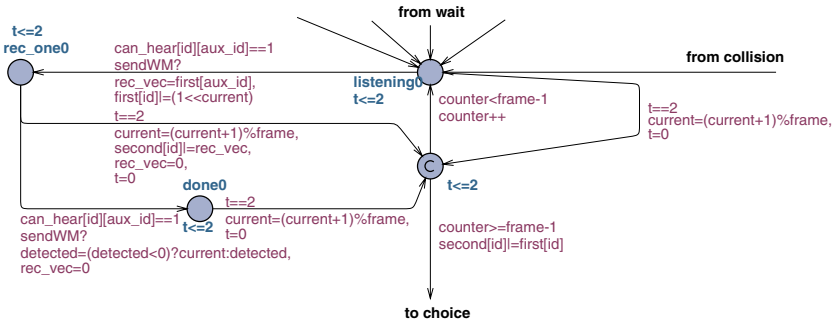

Fig. 4. Model of the discover phase

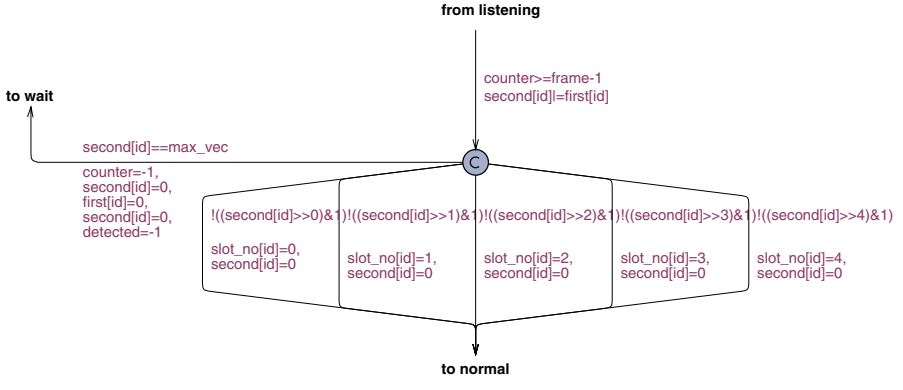

Fig. 5. Model of the choice

most 2 time units, enforced by invariant `t<=2`. If it receives a message from a neighbouring node, it will record the neighbour information of that neighbour (`rec_vec=first[aux_id]`). The node sets the bit for the current slot in its own neighbourhood vector to true (`first[id]|=1<<current`). If the node does not receive any message by the end of the slot (`t==2`), it will increment the current slot number, and move to a committed location.

When the node received one message, it waits in location `rec_one0` either until it receives a second message (collision), or until the end of the slot (`t==2`). The node uses the received neighbourhood information only in the latter case to update the information on slots occupied by the second-order neighbours (`second[id]|=rec_vec`). In the first case the node records if a collision occurred if it was the first collision since the beginning of the discover phase (`detected=(detected<0)?current:detected`). Note, that `detected` has value -1 if no collision has been detected yet. At the end of a slot(`t==2`) the node enters the committed location. If it listened for less than a frame length, it will return to `listening0`, otherwise it will choose a slot.

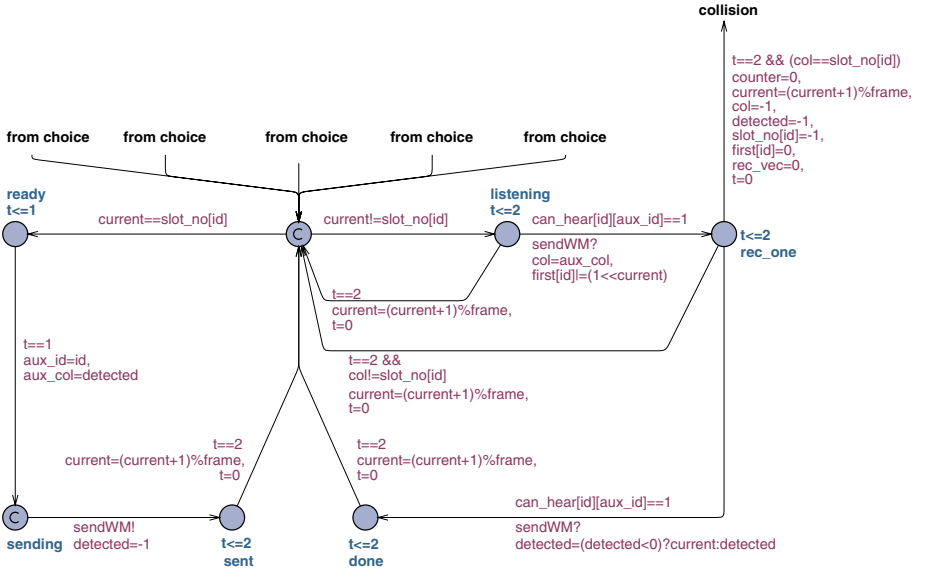

Fig. 6. Model of the active phase

Choosing. Choosing is not a actual phase, but an important intermediate state. Choosing a slot is modelled by a single committed location (Figure 5). Before entering this location the node computes the slots that are neither occupied by the (first-order) neighbours, nor by the second-order neighbours (`second[id] |= first[id]`). If all slots are reported occupied, the node returns to the wait phase (`second[id]==max_vec`)[1]. If there are available slots, i.e the corresponding bits in the bit-vector `second[id]` are equal to zero, the node will select non-deterministically one of these slots.

Active Phase. The main phase of a node is the active phase. The model for this phase is depicted in Figure 6. Locations `ready`, `sending`, and `sent` deal with the transmission of a message, locations `listening`, `rec_one`, and `done` deal with receiving messages.

From the central committed location, which is entered at the beginning of a slot, the node proceeds to send, if the chosen slot number is equal to current slot number (`current==slot_no[id]`), and proceeds to the discover phase otherwise (`current!=slot_no[id]`).

If a node wants to send it waits for one time unit in location `ready`. After one time unit, the node first copies its ID and collision information into global buffers `aux_id`, `aux_col`, and then triggers all nodes in it neighbourhood to update their local information through broadcast channel `sendWM!`. The node then stays in location `sent` until the end of the slot.

[1] Constant `max_vec` is a bit-vector where all elements are set to true.

If a node is ready to receive a message it waits in location `listening`. It remains in that location either until the end of the slot, or until it receives a message. In the former case it increments the slot number at the end of the slot, and proceed with the next slot. In the latter case, if it receives a message, it updates its local information and enter location `rec_one`. If a second message arrives while in `rec_one`, it discards the received information, records the collision (`detected=(detected<0)?current:detected`), and waits for the remaining time of the slot in `done`. If no collision occurred while in `rec_one`, the node proceeds at the end of the slot (`t==2`) depending on the received collision information `col`. If a collision has been reported and it is equal to its slot number (`col==slot_no[id]`), the node returns to the discover phase, and resets all local information. Otherwise, it updates its neighbourhood information, and proceeds with the next slot.

The next section briefly discusses some properties of the timed automaton model of the LMAC protocol, in particular a property that ensures that after a collision nodes involved will choose a new slot.

4.2 Properties

The timed automata model of the LMAC protocol should guarantee basic safety properties. The most basic property is freedom from deadlocks, which can be checked in Uppaal by verifying the following:

$$AG\neg deadlock \tag{1}$$

In addition, we require that the model successfully implements synchronisation of nodes. First, nodes should be synchronised halfway the duration of a slot, since at this time they will send and receive information. We prove for each pair (i, j) of first-order neighbours

$$AG(node_i.t == 1 \Rightarrow node_j.t == 1) \tag{2}$$

In addition neighbours should agree on the current slot number, to ensure that received information is interpreted correctly.

$$AG(node_i.t == 1 \Rightarrow node_i.current == node_j.current) \tag{3}$$

Since we only consider completely connected networks, pairwise synchronisation implies synchronisation of the entire network. The nodes do not to be synchronised when $node_i.t \neq 1$. This can happen when one node increments its current slot number before the other.

In addition to these safety properties the protocol should satisfy a very basic reachability property: There should exist a path to a state, such that all nodes are active, and such that they have a chosen a slot number that is distinct from their first and second-order neighbour's slot. Let $\mathcal{N}$ be the set of all pairs of first and second-order neighbours. We then verify

$$EF \bigwedge_{(i,j)\in\mathcal{N}} (slot\_no(i) \neq slot\_no(j) \wedge active(i) \wedge active(j)) \tag{4}$$

where $active(i)$ is true if a node is its active phase. If the model cannot satisfy this property, it is not even possible to reach a configuration without collision, i.e the related colouring problem has no solution.

The previous property guarantees that there exists a solution, but it does not guarantee that the protocol find this solution. The LMAC protocol chooses slots randomly from the available slots. This is implemented in the timed automaton model as a non-deterministic choice. It is therefore possible that two nodes will repeatedly choose the same slot. For a probabilistic model we could try to prove that with probability one distinct slots will eventually be chosen. Unfortunately, we cannot use the timed automaton model to prove this directly.

Alternatively, we verify two liveness properties to show that the protocol will eventually resolve all conflicts, if satisfied. The first is to show that whenever two first or second-order neighbours choose the same slot number, they will eventually choose a new slot number. We show for each pair (i, j) in $\mathcal{N}$

$$AG\ (slot\_no(i) == slot\_no(j) \wedge sending(i) \wedge sending(j)) \Rightarrow AF(\neg active(i) \vee \neg active(j)) \tag{5}$$

A node may leave the active phase eventually due to a third node reporting the collision or a triggered timeout.

The second liveness property is, that if a node is about to choose a slot, and if it can only choose from one available slot, its neighbours who are in the discover phase are not forced to the make the same choice. The neighbour should eventually be able to choose a different slot. The latter requirement can be dropped, if the neighbour that was forced to a choice, left the active phase and either waits or discovers. For all pairs (i, j) in $\mathcal{N}$ we show

$$AG\ (choosing(i) \wedge available\_slot(i) == 1 \wedge discover(j)) \Rightarrow AF(choosing(j) \wedge (slot\_no(i) \neq slot\_no(j) \vee wait(i) \vee discover(i))) \tag{6}$$

This means that, even if a node is forced to a certain choice ($choosing(i) \wedge available\_slot(i) == 1$), neighbours can eventually choose a different slot.

4.3 Simplification

The model described in Section 4.1 was close to the informal description of the protocol as presented in Section 2. As such each node was equipped with its own clock, and its internal actions completely independent from other nodes.

Checking the reachability probability property (4) was easy, and checking the safety properties (1) to (3) was possible, although demanding in terms of memory and time constraints, while proving the liveness properties (5) and (6) turned out to exceed the memory and time constraints for most topologies. To be able to verify the protocol for all topologies with up to 5 nodes for all properties, we had to simplify the model. The simplification reduced the number of clocks and non-essential interleaving, while keeping the essential behavior.

The simplification builds on two observations. Firstly, that all clocks are synchronised, and secondly that all updates are local. We introduce a scheduler,

with its own clock, that synchronizes the internal update of the nodes at the end of a slot. Without loss of subsequent behavior this scheduler realises a local partial order reduction.

Given that the local clocks of the nodes are only reset during the update of a node, and given that we can safely synchronize all updates, as mentioned before, we find that all clocks are now perfectly synchronised. This means that for clocks $\mathtt{t}_1$ and $\mathtt{t}_2$ holds the invariant $\mathtt{t}_1 == \mathtt{t}_2$. We can therefore safely replace the local clocks of the nodes by the single clock of the scheduler.

The simplification reduced number of clocks and manually introduced a partial order reduction on internal transitions. It should be noted that the scheduler added to the model to achieve this reduction has no equivalent in the actual LMAC protocol. It was purely introduced to reduce the complexity of the model checking problem. If anything it reflects that the LMAC protocol builds on an existing time synchronisation.

5 Results

This section reports on the model checking results for the properties defined in Section 4.2.

While the safety and reachability properties should be satisfied by all models, it is known beforehand that the LMAC protocol is not able to resolve all collisions. This is the subject of the first liveness property (5). Two neighbouring nodes will remain in a collision perpetually, if no third node is able to report this collision, either because there is no third node, or because the third node is unable to send a message without collision. This is a fundamental shortcoming of collision detection algorithms. The aim of the model checking experiments is to iteratively improve the model, and thus the protocol, to reduce the number of topologies that suffer from this problem. This means to reduce the number of topologies and pairs of neighbours that do not satisfy property (5). The improvements deal with modelling bugs, clarification of an ambiguous informal protocol description, to improvements of the protocol.

The model checking experiments have been performed on a Mac Pro with 2 x 3 GHz Dual-Core processor, and a 4 GB 667 MHz memory. We used Uppaal version 4.0. Checking property (5) for a five-node model, i.e. ca. 500 runs of the model checker, took about an hour. This machine outperformed different other PCs, the weakest ones taking a week for the same set of verifications without solving them all, or the better ones, doing the job in a few hours, but still failing due to memory limitations for some experiments, which had to be killed when using too much memory.

5.1 Safety and Reachability Properties

For basic model we assume a network of 4 nodes, and a frame length of 5 slots. For this basic model there are 11 topologies, with 64 pairs of first and second-order neighbours. The experiments show that the basic model (and all models

model 1, topology 4, pair (0,2)

node																																			
0	**0**	0	0	0	0	**0**	***0***	0	0	0	**0**	0	0	0	0	**0**	0	***0***	0	0	**0**	0	**0**	d	d	d	d	**d**	***3***	3	3	3	**3**	***3***	3
1	**i**	d	d	d	d	**d**	**1**	1	1	1	**1**	d	d	d	d	**d**	2	**2**	2	2	***2***	2	**2**	2	2	2	2	**2**	**2**	**2**	2	2	**2**	**2**	**2**
2	**i**	d	d	d	d	**d**	**1**	1	1	1	**1**	d	d	d	d	**d**	2	**2**	2	2	**2**	d	d	d	d	d	3	3	***3***	3	3	3	3	***3***	3
3	i	i	i	i	i	i	**i**	w	w	w	w	w	d	d	d	d	d	**0**	0	0	**0**	0	**0**	d	d	d	d	**d**	4	**4**	4	4	**4**	4	**4**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34

Fig. 7. Scenario of an unresolved collision between node 0 and 2. The y-axis shows the different nodes, the x-axis the time. Each slot contains whether the node is in the initialisation (i), waiting (w), discover (d), or active phase. In the latter case the current slot number is shown. White bold face indicates that the node is sending, black bold face that a node is receiving. Bold italics on a dark (red) background indicate collisions.

that will be derived in the process) satisfy the safety and reachability properties (1) to (4). This means that the models are deadlock free, that the nodes are synchronised, and that for each topology there exist a path that assigns the slots without collision, i.e. that there exists a solution of the related graph colouring problem.

5.2 Liveness Properties

The main liveness property (5) deals with unresolved collisions. In the basic model unresolved collisions may occur for in 3 topologies, for a total of 6 pairs of neighbours. From this basic model for 4 nodes we arrive in 12 iterations at a model that satisfactory resolves collisions for topologies with 5 nodes.

Model 1. This is the basic model for 4 nodes, and a frame length of 5 slots. Among the collisions that are not resolved are collisions that separate a node from the other nodes. An example scenario of such behavior is depicted in Figure 7. It belongs to topology 4, depicted in Figure 8.

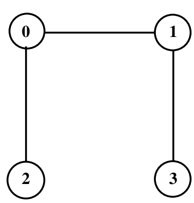

Fig. 8. Topology 4

At time 0 the gateway, node 0, sends a first message. This message is received by node 1 and 2 and they start listening to the medium. One frame later node 1 and 2 both select slot 1, and send at time 6. This leads to a collision at node 0. Node 0 reports the collision at time 10, and node 1 and 2 return to the discover phase. At the end of the scenario node 0 and node 2 collide, perpetually, since there is no neighbour to witness the collision. Node 2 does not receive any message from then on, since it cannot listen while sending. Node 2 entered this collision, because it chose a slot, while it had insufficient information. Node 2 listened from time 21 to time 26, but received not a single message. It had no information about its neighbours, when it made its choice, and any choice had the potential to lead to a collision.

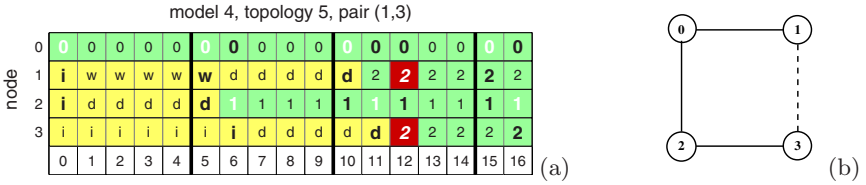

Fig. 9. (a) Scenario of an unresolved collision between node 1 and 3 in topology 5. (b) Topology 5. Node 1 and 3 may fail to resolve a collision.

Model 2. The second model improves on the first model, by introducing the rule that a node may not choose if it received no information in the discover phase. This additional rule successfully deals with the collision depicted in Figure 7.

This model run into problems because it does not reset its first-order neighbour information. After a few repeated choices some node assume that all slots are occupied. They cannot enter the active phase, and consequently cannot report collisions between other nodes. This bug was in the model because of an incomplete informal specification.

Model 3. Model 3 improves on model 2, in that it resets all neighbourhood information after it sends a message. It propagates in the active phase only information collected during the last frame length of slots.

The additional rules in Model 2 and 3 do not eliminate the possibility that a nodes may become disconnected from the network. It may still happen if a node only receives messages while it sends, and no third node witnesses or reports the collision.

Model 4. The fourth model improves on the third model in that a node chooses anew if it does not receive any message in a frame length. This last additional rule resolves all remaining collisions for topologies with 4 nodes which are not ring topology bugs. There is one ring topology, and only two pairs of nodes in it are affected. A scenario leading to this bug is depicted in Figure 9. This kind of collision is however not problematic, since all nodes are able to communicate with the gateway.

Model 5. The fifth model is identical to Model 4, except that it is instantiated for topologies with 5 nodes. There are 61 different topologies, with 571 pairs of neighbours. Although Model 4 was able to resolve all collisions except for the ring topology bug, applied to topologies of 5 nodes many other unresolved collisions suddenly occur. Model checking revealed 56 unresolved collisions, affecting 18 topologies. Also, the model checker was not able to complete for 26 topologies due to memory and time constraints. Once the computer starts swapping memory, progress typically stalls.

Model 6. The sixth model improves on the fifth model by an additional rule. If a node has chosen a slot, and it is active, but has not sent its first message yet,

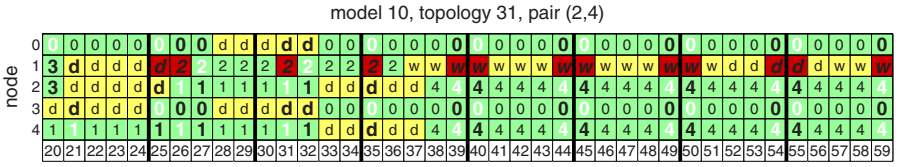

Fig. 10. Scenario of an unresolved collision between node 0 and 3

and if it then receives from a neighbour information that it slot is occupied by a second-order neighbour, then the node proceeds to choose a new slot.

Model 7. The seventh model modifies a rule introduced in Model 2. If it receives in the listing phase only collisions, it does not have sufficient information about its second-order neighbours to make a choice that avoids collisions. The new rule states that a node will not choose if it did not receive a single message, except for collisions.

In the seventh model the following could occur. First, a node reported a collision to all neighbours. Next, these neighbours proceeded to the discover phase. As a consequence, the node which reported the collision would receive no message for a frame length of slots, and incorrectly conclude that it is was disconnected from the network.

Model 8. Model 8 modifies a rule, which was introduced earlier, to avoid the scenario described for model 7. A node concludes that it is alone if it does not hear a neighbour in **two** frame lengths. This prevents a node that reported a collision to conclude that it is disconnected, just because its neighbours went to the discover phase for one frame length.

Model 9. Model 9 further refines the rule about when nodes conclude that they are alone and disconnected. If a node is active, but has not sent yet, it concludes that it disconnected if it has received no message in the frame length of slots right before its first transmission.

Model 10. Model 10 fixes a problem that occurs right after choosing a slot. Model 3 introduced that neighbour information is reset once in a frame length of slots during the active phase. When a mode transitions from the discover phase to the active phase it does not reset the neighbourhood information. As a consequence it may reflect the state of up to two frames length in the past by the time a node is sending. Model 10 fixes this by resetting all information, even if collected during the discover phase, after one frame length. In addition a node concludes that is alone if it hears nothing but collisions for two frame lengths.

Model 11. The eleventh model also refines the rules about when a node has to conclude that it is in a collision. It tackles the problem depicted in Figure 10. Nodes 0 and 3 enter a perpetual collision, since node 1 wrongly concluded at time

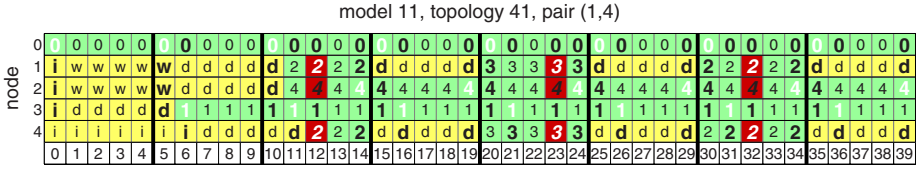

Fig. 11. Node 1 and 4 are perpetually forced to make the same choice

36 that it was disconnected. Node 1 assumed to be alone, since it only heard collisions for two frame lengths. However, the collisions in the frame running from time 27 to 31 differ from the collisions between 33 and 36. Node 1 is not disconnected, and it actually successfully reported a collision at time 32.

Model 11 introduces a new rule about when to conclude that it is in a collision. A node chooses anew if it either receives nothing for two frames or if it witnesses the same collision for the second time. The rational for the latter case is, that if a node observes a collision for the second time, it apparently unsuccessfully reported the collision, likely because it is in a collision itself.

Model 11 resolves all remaining perpetual collisions that happen not in ring topology. The remaining perpetual collisions happen in the ring of 5 nodes, or topologies that contain a ring of 4 nodes. Overall, this are 35 pairs of nodes in 13 topologies that potentially end up in an perpetual collision. These are depicted in Figure 12.

As it comes to the second liveness property – that if a node is forced to choose a slot, all nodes in the discover phase will eventually be able to choose a different slot – it turns out that Model 11 fails for 42 pairs in 14 topologies. Figure 11 depicts an example scenario. First node 1 and 4 both choose the slot 2. This collision is reported at time 14 by node 2. At time 15 node 0 sends its neighbourhood information to node 1. Based on information collected in the frame from time 10 to 14, it reports that all slots but slot 3 are occupied. Node 1 hence has to choose slot 3 at time 19. Node 4 receives in its discover phase messages in slot 1 and 4. In slot 1, it also learns from node 3 that slots 2 and 0 are occupied. Hence, node 4 has to choose node 3 as well, leading to a collision at time 23. This collision gets reported at time 24.

During the next discover phase, both, node 1 and 4 learn that all but slot 2 are occupied. Node 1 and 4 have therefore to choose slot 2 at the end of their discover phase. They end up in a collision again, which gets reported, and at the end of the next discover phase they both have to choose slot 3 again. Etcetera.

Model 12. Model 12 is identical to model 11 except that it assumes a frame length of 6 slots. Increasing the frame size does not influence the number of potential collisions in ring topologies. However, since it increases the number of available slots, all pairs in all topologies now satisfy the second liveness property. If one node is forced to choose a certain slot, the second can eventually choose a slot that differs from the first nodes slot.

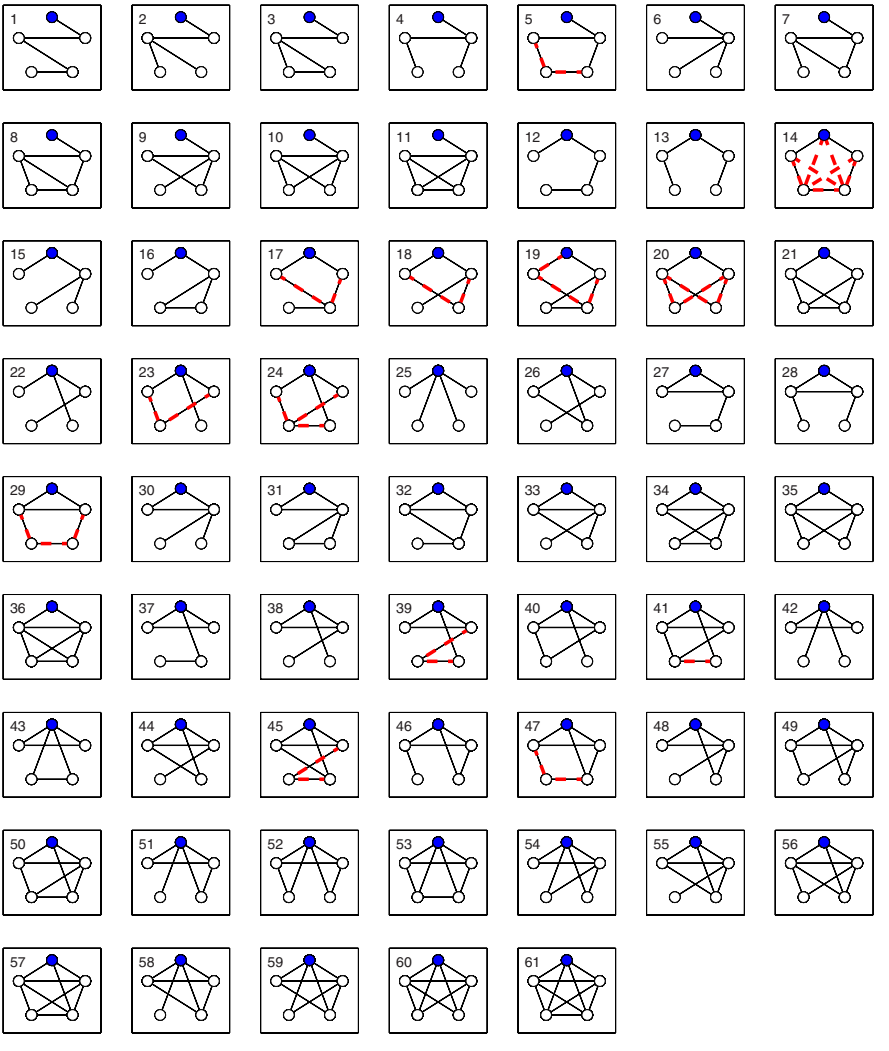

Fig. 12. Final results for all 61 topologies with 5 nodes. The gateway is the solid node. Dashed lines depict pairs of neighbours that **may** end up in an perpetual collision. Only in the ring topology 14 this may happen also between second-order neighbours.

6 Conclusion

In this paper we reported about the analysis of a medium access protocol for wireless networks, the LMAC protocol. The analysis technique we applied was model checking, using the timed automaton model checker Uppaal [3]. Our approach was a systematic analysis of all possible connected network topologies with 4 and with 5 nodes. The most relevant property we investigated was, whether

collisions are detected and a new choice of slots is initiated afterwards. We checked 12 different models, four for all topologies of four nodes, eight for all topologies consisting of five nodes. The sequence of models reflects the increments in insight in the protocol, and in the improvements of the protocol. Figure 12 shows the results for the last of the models.

Checking the models against a number of properties summed up to more than 8000 model checking runs in total. For example, in each of the eight five node models there are 571 pairs of nodes. For each pair it needs to be investigate whether a possible collision is detected by the protocol or not. This results in 4568 instances of property 5 alone that need to be model checked.

Extending the systematic analysis to 6 node topologies would not only increase the model checking time for each instance, but also the number of instances to investigate. With 6 nodes we would have 486 different topologies and 6273 pairs of nodes to analyse. This would lead not only to a state space explosion problem within one model, but to a much higher extent to a instance explosion problem. For the state space explosion fully symbolic model checking techniques could be helpful, but not for the instance explosion problem. Furthermore, it seems to be difficult to parameterise topologies, having parametric model checking techniques in mind. An alternative approach for showing correctness for a class of topologies, using a combination of model-checking and abstract interpretation, was presented in [1]. Here however, we face the additional problem that essential properties are not valid for a number of instances. Therefore, we argue that with straightforward model checking techniques, not much more can be done. A possible extension could be stochastic analysis with a probabilistic model checker, which will be discussed below.

There are three main results: (1) the description of the protocol is improved, (2) the protocol itself is improved, and (3), problematic topologies with possible scenarios of unresolved collision have been identified.

Improvement of the protocol description. We had a quite usual experience here: several "bugs" found in first rounds of analysis turned out to be present in the documentation of the protocol, but not in the implementation. The respective "patches" were added to the documentation.

Protocol improvements. Some scenarios leading to unresolved collisions helped to improve the protocol, and were absent in the later protocol versions:

- There is an additional trigger for the choice of a new slot: if a node hears nothing, it concludes that it is isolated or participating itself in an collision, and starts a new choice.
- If a node hears the same collision twice, it concludes that its collision report has not been heard. The only reason for this is that this node itself is in a collision. Therefore it starts a new choice in this situation.
- Some situations of collision detected could be solved by a change in parameters in the protocol, e.g., the time that a node listens before it chooses a new slot, was extended from one frame to two frames.

– The frequency of information update was increased, e.g. slots where collisions were heard are only stored for one frame. Timely resets seem to be crucial for the protocol.

Protocol faults. It is the case that collisions are not detected if there is not a third node which can observe the collision. This situation occurs in all topologies containing a square. Fortunately, even when there is a collision, all nodes are still connected to the gateway, which makes these collisions less dramatic. The only exception to this pattern is the ring-topology of five nodes, where also unresolved collision can occur.

As mentioned, the colouring problem that the LMACprotocol tries to solve is NP-hard. It cannot be expected that a light-weight, distributed algorithm finds a solution in all cases.

Further results are:

Justification of the verification approach. The real faults found in the protocol were detected in non-trivial scenarios, generated by Uppaal-counterexamples and, for readability, transformed to a graphic by a Matlab procedure. Figure 10 contains an example of such a scenario. It is obvious that these scenarios, due to complexity, are unlikely to be found during a simulation run.

Justification of the analysis of *all* possible topologies. We found that small changes in the topology can lead to different results. Intuitively, one would expect that "similar" topologies give similar results. Unfortunately, any intuition of this kind was proved wrong. Also another intuition, that most collisions occur when the connectivity is higher turned out to be wrong. It turns out the collisions get resolved when the connectivity is high. This justifies our approach of systematically investigating all topologies. Selecting "representative" topologies is misleading, because there are no criteria for what "representative" could be.

Quantification of the success rate. For the 61 topologies we investigated 571 pairs of nodes for collision detection. 35 pairs of these showed a possible unresolved collision. There are two aspects of probability present: first, for a fixed topology we could determine the probability of an undetected collision. This exceeds the possibilities of Uppaal, and would require a probabilistic model checker (what we have not done). The second aspect is the probability of a certain topology. This cannot be answered in general, because it depends on the application domain, and the level of mobility in the network investigated.

Future work. We have not considered the probabilistic aspects of the protocol. There are two sources of probabilities in the protocol: the choice of a new slot out of all free slots, and the waiting time before choosing a new slot. We see two different approaches to treat these aspects: one is by simple meta-argumentation, based on combinatorics and elementary stochastics (e.g.,"What is the probability that two nodes keep choosing the same waiting times?"). The other possibility is by using a probabilistic model checker, like PRISM. However, probabilistic models are typically even more complex than the ones we considered, which

decreases the limit of what can be analysed. In this case a number of effective abstraction steps have to be applied to the model, to decrease its complexity.

We have not yet considered aspects of energy efficiency in the choice of new slots. One source of energy consumption is the number of iterations are necessary, to choose a slot without creating a collision. To answer this question probabilistic analysis is necessary. Another source of energy consumption is in the number of hops that a packet needs to reach the gateway. The choice of a slot can influence latency. Here, it seems that the "more deterministic" choice for a latency-minimizing slot increases the chance for collision during the slot selection phase. In contrary, when we apply a uniformly distributed choice of slots during the selection phase, the latency will not be optimal. What the right balance is between these parameters is subject to further analysis.

References

1. Bauer, J., Schaefer, I., Toben, T., Westphal, B.: Specification and verification of dynamic communication systems. In: Application of Concurrency to System Design (ACSD'06), pp. 189–200. IEEE Computer Society, Los Alamitos (2006)
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems: SFM-RT 2004. LNCS, vol. 3185, Springer, Heidelberg (2004)
3. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Quantitative Evaluation of Systems - (QEST'06), pp. 125–126. IEEE Computer Society Press, Los Alamitos (2006)
4. Brinksma, E.: Verification is experimentation! Int. J. on Software Tools for Technology Transfer 3(2), 107–111 (2001)
5. Cardell-Oliver, R.: Why Flooding is Unreliable (Extended Version). Technical Report UWA-CSSE-04-001, CSSE, University of Western Australia (2004)
6. Mader, A., Wupper, H., Boon, M.: The construction of verification models for embedded systems. Technical report TR-CTIT-07-02, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands (January 2007)
7. Moscibroda, T., Wattenhofer, R.: Coloring unstructured radio networks. In: Proc. of 17th Symposium on Parallelism in Algorithms and Architectures (2005)
8. Olveczky, P., Thorvaldsen, S.: Formal modeling and analysis of wireless sensor network algorithms in real-time maude. In: Proceedings of the 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2006), IEEE Computer Society Press, Los Alamitos (2006)
9. Sridharan, A., Krishnamachari, B.: Max-min fair collision-free scheduling for wireless sensor networks. In: Workshop on multi-hop wireless networks (2004)
10. van Hoesel, L.F.W., Havinga, P.J.M.: A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In: In 1st International Workshop on Networked Sensing Systems (INSS 2004), pp. 205–208 (June 2004)