

Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks

Ansgar Fehnker¹, Lodewijk van Hoesel², Angelika Mader^{2*}

¹ National ICT Australia** and University of New South Wales, Australia

² Department of Computer Science, University of Twente, The Netherlands
ansgar.fehnker@nicta.com.au, l.f.w.vanhoesel@utwente.nl, mader@ewi.utwente.nl

Abstract. In this paper we report about 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. To evaluate this property for all connected topologies more than 8000 model checking runs were required. Increasing the number of nodes would not lead only to state space problem, but to much more 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 localized strategy of choosing a time slot for nodes.

The basic idea of the protocol is quite simple. However, due to distribution and a number of parameters, the possible behaviours get 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

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

of an implementation, but not 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 a number of instances of different topologies.

Model checking as a way to increase the confidence comes also into play, as we do is not aim to prove that the protocol is correct for all topologies. This in contrast 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 a 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 [3, 2]. There are the following reasons for the choice of the model checking approach considering all topologies:

(1) We believe that relevant faults appear already in networks with a small number of nodes. Of course, possible faults that involve higher numbers of 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 [8] considering 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 was 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. Furthermore, the explicitness of modelling decisions makes it easier to interpret the result of the model checking experiments, i.e. to identify what was proven, and what not. The reader not interested in the details of the model should skip therefore section 4. Other readers will find there information for the reconstruction of the model.

The goal of the protocol is to find a mapping of time slots to nodes avoiding collision when sending messages is avoided. 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.

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

2 The LMAC protocol

In schedule-based MAC protocols, time is organized 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 synchronization.

The control message of the LMAC protocol plays an important role in obtaining a local two-hop view of the network. With each transmission a node broadcasts a bit vector of slots occupied by its (first-order) neighbours and itself. When a node receives a message from a neighbour, it marks the respective time slots as occupied. To maintain synchronization 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. 2):

Initialisation phase (I) — The node samples the wireless medium (at a low rate to conserve energy) to detect other nodes. When a neighbouring node is detected, the node synchronizes (i.e. the node knows the current slot number). When a new frame is due, the node switches to the wait phase W .

Wait phase (W) — We observed that especially at network setup, many nodes receive an impulse to synchronize at the same time. We introduce randomness in reaction time between synchronising with the network and actually choosing of a free time slot. After the random wait time, the node continues with 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 do not interfere in its second order neighbourhood and can be used freely. At this moment the set of

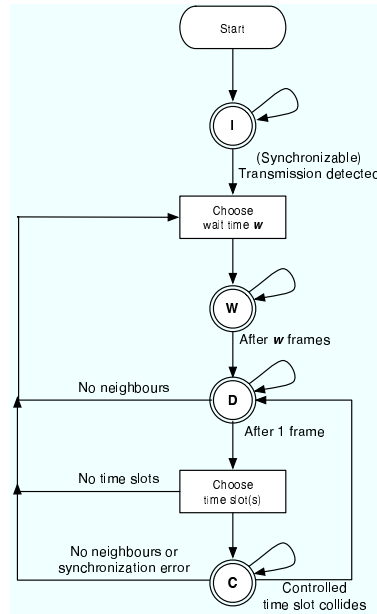


Fig. 1. Control flow diagram of the protocol

non-interfering time slots is available. Note that the node can choose *any* time slot of this set to control. To reduce the probability of collisions (i.e. two or more nodes that claim equal time slots and are interfering with each other), we let nodes randomly choose one from the set of available slots.

Active phase (A) — The node transmits a message in its own time slot. Meanwhile it listens to 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 the node continues proceeds 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 (i.e. many nodes wake-up at same time) or when network topology changes due to mobility of nodes.

The nodes that caused the collision cannot detect the collision by themselves; they need to be informed by their neighbouring nodes, simply because they are transmitting when the event occurs. 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 fall back to the wait phase W .

3 Timed automata

Systems are modelled in Uppaal as a parallel composition of non-deterministic, 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 the various locations in the automata and the invariants in the various locations may contain both integer-valued variables and real-valued clocks. Clocks may also be reset to some constant in the transitions. Several automata may also synchronize on transitions using handshakes. With the use of shared variables it is possible to model data transfer between automata. Locations may be urgent, which means time is not allowed to progress, and 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 CTL.

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 5 phases, following the states of the protocol as shown in figure 2. The first part is the initialisation phase, the second the optional wait phase. The next part models

the discover phase which gathers neighbourhood information. The fourth phase is to choose a slot, and the fifth and last phase is 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 base line 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 also has five local variables. Variable `rec_vec` is a local copy of 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 `collision` 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. 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 sent. 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 a message. This synchronisation is the subject of one of the properties that will be verified in the remainder.

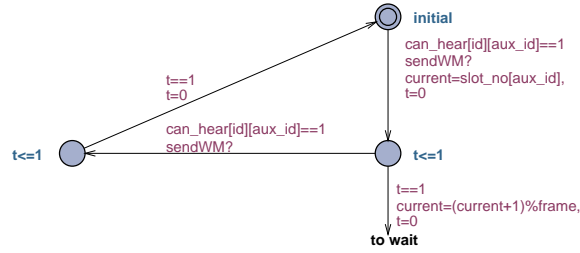


Fig. 2. Model of the initialisation phase

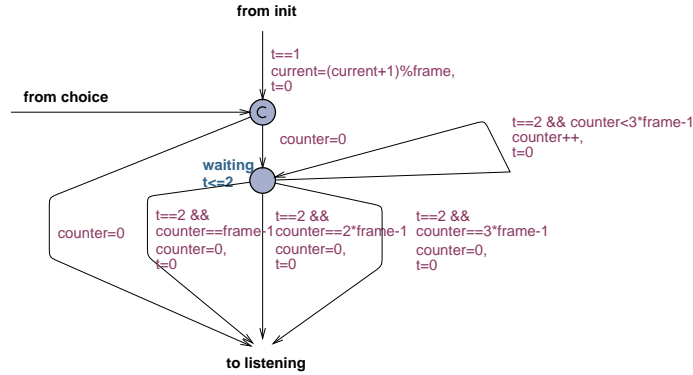


Fig. 3. Model of the wait phase

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 location. If no collision occurs, the node will proceed to the next slot, increment the current slot counter modulo the length of the frame ($\text{current}=\text{current}+1\% \text{frame}$), 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 $\text{counter}<3*\text{counter}-1$. 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.

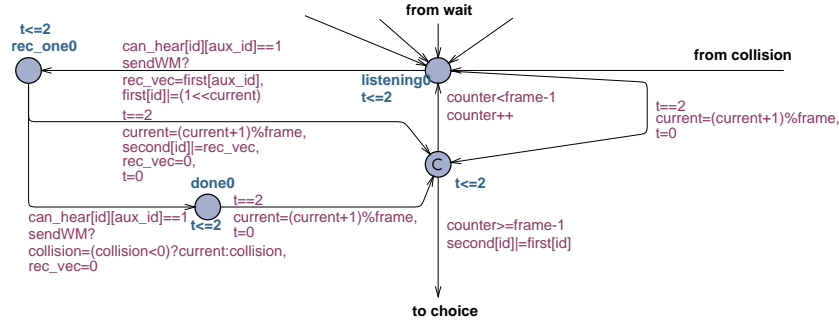


Fig. 4. Model of the discover phase

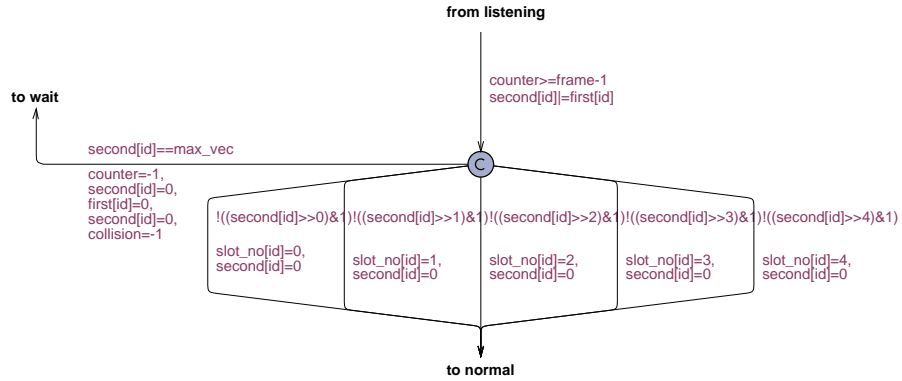


Fig. 5. Model of the choice.

Clocks and variables will be updated as follows. When a node enters node **listening0**, the local clock will be zero. It will wait in this location for at most 2 time units, enforced by invariant $t \leq 2$. If it receives a message from a neighbouring node, it will record the neighbour information of that neighbour ($\text{rec_vec} = \text{first}[\text{aux_id}]$). The node sets the bit for the current slot in its own neighbourhood vector to true ($\text{first}[id] \mid= 1 \ll \text{current}$). 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 ($\text{second}[id] \mid= \text{rec_vec}$). In the first case the node records if a collision occurred if it was the first collision since the beginning of the discover phase ($\text{collision} = (\text{collision} < 0) ? \text{current} : \text{collision}$). Note, that **collision** has value -1 if no collision has been reported yet. At the end of a slot ($t = 2$) the

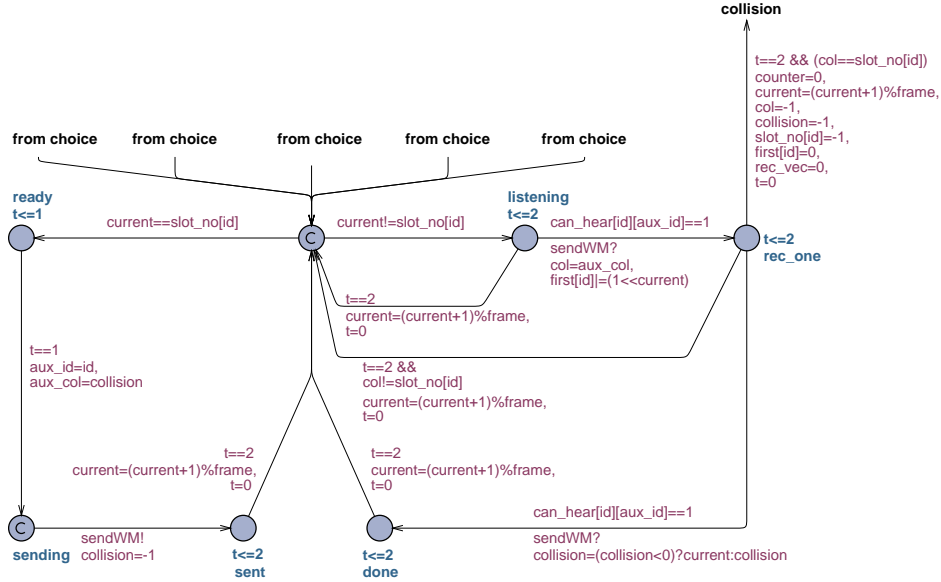


Fig. 6. Model of the active phase.

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.

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`). 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.

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 (`collision=(collision<0)?current:collision`), 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:

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

In addition, we require that the model successfully implements synchronization of nodes. First, nodes should be synchronized 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

$$AG(\text{node}_i.t = 1 \Rightarrow \text{node}_j.t = 1) \tag{2}$$

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

$$AG(\text{node}_i.t = 1 \Rightarrow \text{node}_i.\text{current} = \text{node}_j.\text{current}) \tag{3}$$

Since, we only consider completely connected networks, pairwise synchronization implies synchronization of the entire network.

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

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

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 coloring problem has no 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, and the timed automaton model cannot be used to prove that with probability one distinct slots will be chosen.

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}

$$\begin{aligned} AG \ (slot_no(i) = slot_no(j) \wedge sending(i) \wedge sending(j)) \\ \Rightarrow AF(\neg active(i) \vee \neg active(j)) \end{aligned} \quad (5)$$

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 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

$$\begin{aligned} AG \ (choosing(i) \wedge available_slot(i) = 1 \wedge discover(j)) \Rightarrow \\ AF(choosing(j) \wedge (slot_no(i) \neq slot_no(j) \vee wait(i) \vee discover(i))) \end{aligned} \quad (6)$$

This ensure that, even if one neighbour was forced into a choice ($choosing(i) \wedge available_slot(i) = 1$), neighbours will be able to 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 synchronized, 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 synchronized. This means that for clocks τ_1 and τ_2 holds the invariant $\tau_1 = \tau_2$. 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 synchronization.

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.

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 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 synchronized, 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 6 pairs of neighbours. These 6 pairs

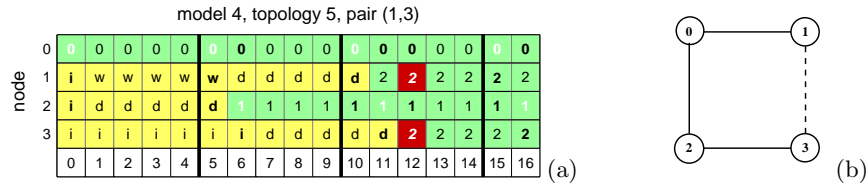


Fig. 8. (a) Scenario of an unresolved collision between node 1 and 3 in topology 5. (b) Topology 5. Node 1 and 3 may fail to resolve a collision.

Model 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 8. 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, and if it then receives from a neighbour information that its 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 an uncorrupted message.

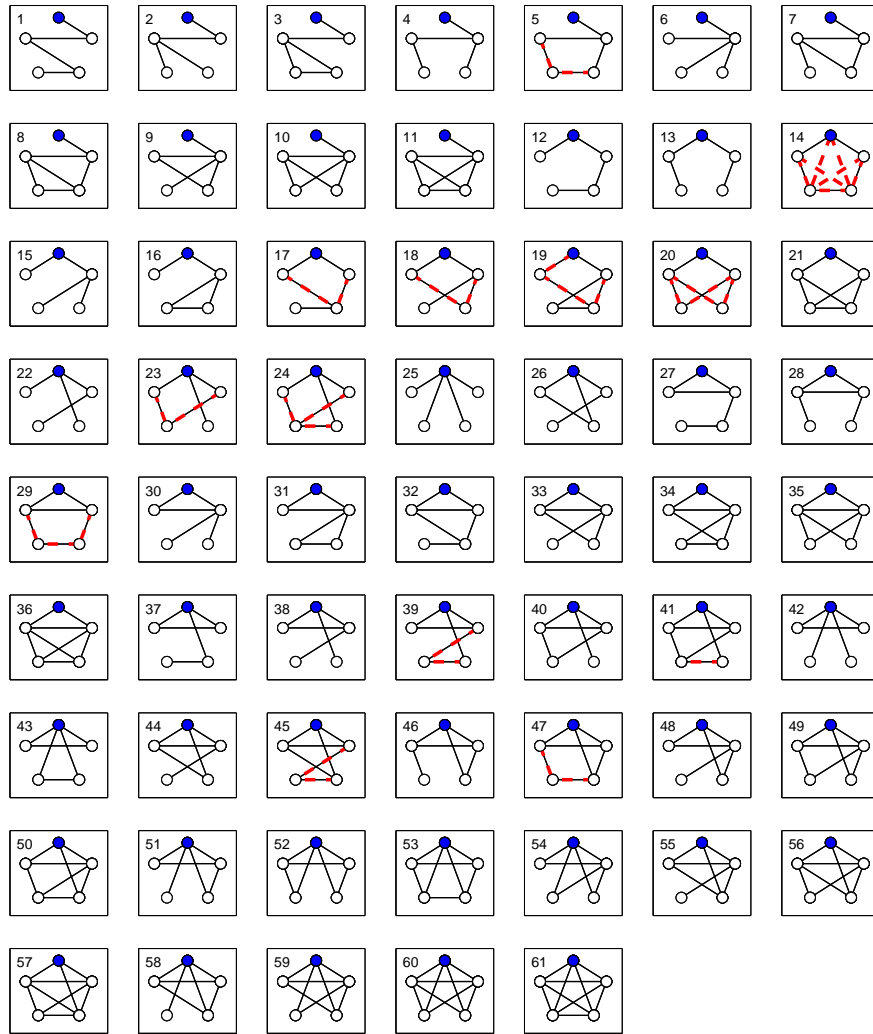


Fig. 11. Final results for all 61 topologies with 5 nodes. The gateway is the solid (blue) node. Dashed (red) lines depict pairs of neighbours that **may** end up in an perpetual collision. Only in the ring topology 14 this may happen also between second order neighbours.

consisting of five nodes. The sequence of models reflects the increments in insight in the protocol, and in the improvements of the protocol. Figure 11 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 the five node model there are

571 pairs of nodes to investigate whether a possible collision is detected by the protocol or not. Extending the systematic analysis to 6 node topologies would not only increase the model checking time for each instance, i.e. each pair of nodes to analyse, 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 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 LMAC protocol 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 9 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 probabilism 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. Jorg Bauer, Ina Schaefer, Tobe Toben, and Bernd Westphal. Specification and verification of dynamic communication systems. In *Application of Concurrency to System Design (ACSD'06)*, pages 189–200. IEEE Computer Society, 2006.
2. G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer-Verlag, September 2004.
3. Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Quantitative Evaluation of Systems - (QEST'06)*, pages 125–126. IEEE Computer Society, 2006.
4. E. Brinksma. Verification is experimentation! *Int. J. on Software Tools for Technology Transfer*, 3(2):107–111, May 2001.
5. R. Cardell-Oliver. Why Flooding is Unreliable (Extended Version). Technical Report UWA-CSSE-04-001, CSSE, University of Western Australia, 2004.
6. A. Mader, H. Wupper, and M. Boon. 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, Jan 2007.
7. T. Moscibroda and R. Wattenhofer. Coloring unstructured radio networks. In *Proc. of 17th Symposium on Parallelism in Algorithms and Architectures*, 2005.
8. P. Olveczky and S. Thorvaldsen. 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, 2006.
9. A. Sridharan and B. Krishnamachari. Max-min fair collision-free scheduling for wireless sensor networks. In *Workshop on multi-hop wireless networks*, 2004.
10. 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 *1st International Workshop on Networked Sensing Systems (INSS 2004)*, pages 205–208, June 2004.