Statistical Model Checking of a Clock Synchronization Protocol for Sensor Networks*

Luca Battisti, Damiano Macedonio, and Massimo Merro

Dipartimento di Informatica, Università degli Studi di Verona, Verona, Italy

Abstract. This paper uses the statistical model checking tool in the UPPAAL toolset to test the robustness of a distributed clock synchronization algorithm for wireless sensor networks (WSN), in the case of lossy communication, i.e., when the WSN is deployed in an environment with significant multi-path propagation, leading to interference. More precisely, the robustness of the gMAC protocol included in the Chess WSN platform is tested on two important classes of regular network topologies: cliques (networks with full connectivity) and small grids (where all nodes have the same degree). The paper extends previous work by Hedaraian et al. that only analyzed this algorithm in the ideal case of non-lossy communication, and only in the case of cliques and line topologies. The main contribution is to show that the original clock synchronization algorithm is not robust to changing the quality of communication between sensors. More precisely, with high probability the algorithm fails to synchronize the nodes when considering lossy communication over cliques of arbitrary size, as well as over small grid topologies.

1 Introduction

Wireless sensor networks (WSNs) are (possibly large-scale) networks of sensor nodes deployed in strategic areas to gather data. Sensor nodes collaborate using wireless communications with an asymmetric many-to-one data transfer model. Typically, they send their sensed data to a sink node which collects the relevant information. WSNs are primarily designed for monitoring environments that humans cannot easily reach (e.g., motion, target tracking, fire detection, chemicals, temperature); they are used as embedded systems (e.g., biomedical sensor engineering, smart homes) or mobile applications (e.g., when attached to robots, soldiers, or vehicles).

In wireless sensor networks, the basic operation is *data fusion*, whereby data from each sensor is agglomerated to form a single meaningful result. The fusion of individual sensor readings is possible only by exchanging messages that are timestamped by each sensor's local clock. This mandates the need for a common notion of time among the sensors which is achieved by means of so called *clock synchronization protocols* [13,15].

* Work partially supported by the PRIN 2010-2011 project "Security Horizons".

F. Arbab and M. Sirjani (Eds.): FSEN 2013, LNCS 8161, pp. 168–182, 2013.
DOI: 10.1007/978-3-642-40213-5_11,

In this paper we do model checking of a distributed algorithm of clock synchronization for WSNs that has been developed by the Dutch company CHESS [12]. In order to realize an energy efficient communication mechanism, CHESS developed a gossip-based MAC algorithm [14] (abbreviated gMAC) which is responsible for regulating the access to the wireless shared channel. Here we are interested in verifying the robustness of the gMAC algorithm in the presence of *packet loss*. Packet loss is particularly relevant in wireless sensor networks which are deployed in environments with significant multi-path distortion (when part of the signal goes to the destination while another part bounces off an obstruction and then goes on to the destination). Most of sensor platforms do not have enough frequency diversity to reject multi-path propagation.

Our work has been strongly inspired by a recent analysis [8] of the gMAC synchronization protocol on clique and line topologies, in the ideal case of non-lossy communication. In the case of line topologies, paper [8] shows that the protocol fails to synchronize all nodes, when the number of nodes grows. On the other hand, on clique topologies the protocol behaves quite well and the paper provides constraints on *guard times* (delays added before and after the transmission of sync messages) to guarantee clock synchronization, independently on the clique size. In [8] the protocol is modeled as a network of timed automata and verified using the UPPAAL model checker [2,3]. However, the model in [8] does not incorporate several features such as dynamic slot allocation, uncertain communication delays, and unreliable radio communication. In the current paper we extend their analysis by adopting a *probabilistic model* of radio communication that takes into account message loss according to the measurement of packet delivery suggested in [17]. As our model is a network of probabilistic timed automata, we decided to do our analysis by applying *Statistical Model Checking* (SMC) [7] within the UPPAAL toolset [2,3]. SMC consists in monitoring a proper number of runs of the system and then applying a statistical algorithm to obtain an estimate of the result of the desired query.

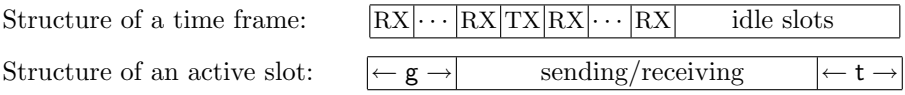
Our analysis shows that low guard times (within the safety range proposed in [8]) are not sufficient to guarantee clock synchronization in clique topologies of arbitrary size. More precisely, in the case of lossy communication, the size of the clique *does* play a crucial role in the effectiveness of the protocol: the bigger is the clique the higher must be the guard time to ensure clock synchronization with high probability. Here it is important to notice that guard times cannot be arbitrary increased without dramatically affecting the duration of the battery life of the sensor nodes [1]. Finally, we move our analysis on grid topologies, with increasing neighbor degree, to better simulate a uniform node distribution of sensor nodes in a given area. Our simulations show that high values of the guard times may be not sufficient to guarantee clock synchronization in the presence of message loss, even in small 5×5 grid networks. On the other hand, we observe that the efficiency of the protocol improves when the number of neighbours, and hence node connectivity, increases.

Outline Section 2 introduces the gMAC protocol. Section 3 illustrates the corresponding UPPAAL probabilistic model. Section 4 details our analysis on cliques

and grid topologies. Section 5 concludes the paper with final remarks, future and related work.

2 The gMAC Protocol

The gMAC protocol is a Time Division Multiple Access (TDMA) protocol, where time is divided into fixed length *frames*, and each frame is subdivided into *slots*. Slots can be either *active* or *idle*. During active slots, a node is either listening for incoming messages from neighbouring nodes (RX slot) or it is sending a message (TX slot). During idle slots a node is switched to energy saving mode. Active slots are gathered in a contiguous sequence placed at the beginning of each frame.



Since energy efficiency is a major concern in the design of wireless sensor networks, the number of active slots is typically much smaller than the total number of slots. In the implementation of gMAC the number of slots within a frame is 1129 out of which 10 are active. A node can only transmit a message once per time frame in its TX slot. If two neighbouring nodes choose the same send slot then a communication collision will occur in the intersection of their transmission ranges preventing message delivery. In the original protocol a node randomly chooses an active slots as *send slot* (TX slot) considering all other active slots as *receive slots* (RX slots). However, for the sake of simplicity, as in [8], in our analysis we assume that the TX slots are fixed and have been chosen in such a way that no collision occurs.

In order to ensure that when a node is sending all its neighbours are listening, *guard times* are introduced. This means that each sender waits for some time (g clock cycles) at the beginning of its TX slot to ensure that all its neighbours are ready to receive messages; similarly, a sender does not transmit for a certain amount of time (t clock cycles) at the end of its TX slot. Guard times cannot be arbitrary increased without dramatically affecting the duration of the battery life of the sensor nodes [1]. So, the choice of proper guard time values is crucial in the protocol design. In the current implementation, each slot consists of 29 clock cycles, out of which 18 cycles are used as guard time.

The CHESS sensor nodes come equipped with a 32 kHz crystal oscillator that drives an internal clock used to determine the beginning and the end of each slot. Sensor nodes are also equipped with an ATMega64 micro-controller and a Nordic nRF24L01 [10] packet radio. Depending on the environment, the Nordic nRF24L01 radio has a transmission range between $0.5m$ and $50m$, For the sake of simplicity we assume that all nodes have the same transmission range; this means that the transmission between nodes is assumed to be symmetric.

3 UPPAAL Probabilistic Model for gMAC

In this section, we provide a small extension of the UPPAAL model for gMAC of [8] in which a probabilistic choice to model message loss is introduced. The model assumes a finite, fixed set of sensor nodes $\mathsf{Nodes} = \{0, \ldots, \mathsf{N}-1\}$. The behaviour of each individual node $i \in \mathsf{Nodes}$ is described by means of three different timed automata: **Clock**(i), **WSN**(i), **Synchronizer**(i). Automaton **Clock**(i) models the hardware clock of the node, **WSN**(i) takes care of sending messages, and **Synchronizer**(i) re-synchronizes the hardware clock upon receipt of a message. The automaton **Synchronizer**(i) is the only one where probabilities are introduced to model packet loss. The complete model consists of the composition of the three automata **Clock**(i), **WSN**(i) and **Synchronizer**(i), for each $i \in \mathsf{Nodes}$.

For each node i there are two state variables: $\mathsf{clk}[i]$, which records the value of the hardware clock (initially 0), and $\mathsf{csn}[i]$, which records the current slot number (initially 0). Furthermore, there are two broadcast channels: $\mathsf{tick}[i]$, used to synchronize the activities within the node i, and $\mathsf{start\_message}[i]$, used to inform all neighbours of the beginning of i's transmission. Table 1 reports the protocol parameters.

Figure 1 depicts the automaton **Clock**(i) of [8] modeling the hardware clock of node i. The local clock variable x measures the time between two consecutive clock ticks. A $\mathsf{tick}[i]!$ action is enabled when x reaches the value min and must fire before x reaches the value max. When the action $\mathsf{tick}[i]!$ occurs the variable x is reset to 0 and the variable $\mathsf{clk}[i]$ is incremented by 1. As explained in [8], the state variable $\mathsf{clk}[i]$ is reset after k_0 clock ticks for the model checking to become feasible. A realistic clock drift rate is about 20 ppm (parts-per-million). Such a rate is achieved in the model by setting $\mathsf{min} = 10^5 - 2$ and $\mathsf{max} = 10^5 + 2$. In the model of [8] ticks may nondeterministically occur within the time interval $[\mathsf{min}, \mathsf{max}]$; thus the delay between two $\mathsf{tick}[i]!$ actions is nondeterministic. The stochastic semantics for timed automata of UPPAAL SMC excludes nondeter-

Table 1. Protocol parameters

Parameter	Description	Constraints
N	number of nodes	$0 < \mathsf{N}$
C	number of slots in a time frame	$0 < \mathsf{C}$
n	number of active slots in a time frame	$0 < \mathsf{n} \leq \mathsf{C}$
k_0	number of clock ticks in a time slot	$0 < \mathsf{k}_0$
$\mathsf{csn}[i]$	current slot number for node i	$0 \leq \mathsf{csn}[i] < \mathsf{n}$
$\mathsf{tsn}[i]$	TX slot number for node i	$0 \leq \mathsf{tsn}[i] < \mathsf{n}$
$\mathsf{clk}[i]$	clock value for node i	$0 \leq \mathsf{clk}[i] < \mathsf{k}_0$
g	guard time	$0 < \mathsf{g}$
t	tail time	$0 < \mathsf{t}$
min	minimal time between two clock ticks	$0 < \mathsf{min}$
max	maximal time between two clock ticks	$\mathsf{min} \leq \mathsf{max}$
loss	*message loss probability*	$0 \leq \mathsf{loss} \leq 100$

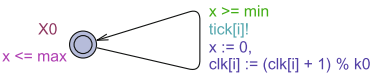


Fig. 1. Clock(i)

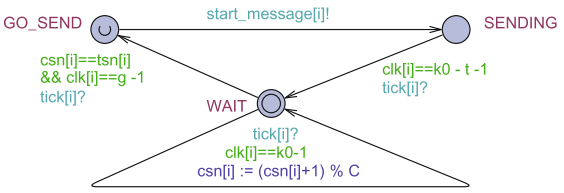


Fig. 2. WSN(i)

minism; thus the delay between two $\mathsf{tick}[i]!$ actions is implemented as a uniformly distributed stochastic delay. This may be non-realistic in general: clocks usually run either too fast or too slow for long periods of time, due to environmental differences. However in our analysis we will focus on small networks and we will assume all sensors being in the same environmental conditions. Thus we exploit UPPAAL SMC's random clock speed jitter.

Figure 2 describes automaton **WSN**(i) of [8] devoted to message sending. The automaton waits in the initial location WAIT until the current slot number $\mathsf{csn}[i]$ equals the TX slot $\mathsf{tsn}[i]$, and g ticks occur in that slot. Then the automaton moves to the location GO_SEND which is immediately left by performing a $\mathsf{start\_message}[i]!$ action. This action leads the automaton to the location SENDING. The automaton remains in this location until the beginning of the tail interval (which starts after $\mathsf{k}_0 - \mathsf{t}$ ticks). Then the automaton returns to the location WAIT where it increments the current slot number $\mathsf{csn}[i]$ every k_0 ticks.

Figure 3 contains the automaton **Synchronizer**(i) which is devoted to synchronize the hardware clock. We enrich the corresponding automaton of [8] with a simulation of message loss on the channel $\mathsf{start\_message}[i]$. The UPPAAL model checker features branching edges with associated weights for the probabilistic extension. Thus we define an integer constant loss, with $0 \leq \mathsf{loss} \leq 100$, and every node can either lose a message with weight loss or receive it with weight $(100 - \mathsf{loss})$. The automaton **Synchronizer**(i) waits in its initial location S0 until it detects, in an active slot ($\mathsf{csn}[i] < \mathsf{n}$), the beginning of a new message from a neighbor j (action $\mathsf{start\_message}[j]?$). When this happens the automaton moves to a committing location C and it immediately goes to a branching edge where: (i) with weight loss it returns in its initial location S0 and (ii) with weight $100 - \mathsf{loss}$ it goes to location S1. Case (i) formalizes the loss of the starting message from node j, while case (ii) formalizes the reception of the same message. Notice that UPPAAL requires input determinism to ensure that the system to be tested always produces the same outputs on any given sequence of inputs. Thus we need an extra intermediate location instead of branching immediately on

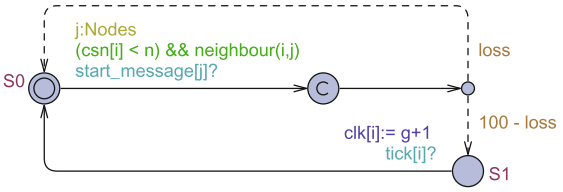


Fig. 3. Synchronizer(i)

the start_message$[j]$! action. Notice also that the start_message$[j]$! action occurs exactly when $\mathsf{clk}[j] = \mathsf{g}$ (as a result of the synchronization of automata **WSN**(j) and **CLOCK**(j)); while the node i, by means of automaton **Synchronizer**(i), resets the variable $\mathsf{clk}[i]$ to $\mathsf{g}+1$ after a further tick. The guard $\mathsf{neighbor}(i,j)$ indicates that i and j are in the transmission range of each other. Formally, neighbor() is a symmetric function related to the slot allocation in the following manner [8]:

$$\begin{aligned} \mathsf{neighbor}(i,j) &\Longrightarrow \mathsf{tsn}[i] \neq \mathsf{tsn}[j] \\ \mathsf{neighbor}(i,j) \wedge \mathsf{neighbor}(i,k) &\Longrightarrow \mathsf{tsn}[j] \neq \mathsf{tsn}[k] \end{aligned} \tag{1}$$

This means that whenever two nodes are neighbours or have a common neighbor then they must have distinct TX slot numbers. The function neighbor() is helpful to provide a formal definition of synchronized sensor networks. Intuitively, a sensor network is said to be synchronized if, whenever a node is sending in a given slot number then all neighbouring nodes are in the same slot number.

Definition 1. *A network is said to be synchronized if for all reachable states* $(\forall i,j \in \mathsf{Nodes})((\mathsf{SENDING}_i \wedge \mathsf{neighbor}(i,j)) \Longrightarrow (\mathsf{csn}[i] = \mathsf{csn}[j]))$.

4 Our Analysis

UPPAAL Statistical Model Checker [7] evaluates properties on execution runs of a network of probabilistic timed automata. The execution time of these runs is represented by a variable `time`. It is left to the user to set a bound to this variable. In particular, fixed a constant value `bound`, the Statistical Model Checker can reply to queries of the following shape

```
Pr[time<=bound] (<> expr)
```

by performing an adequate number of runs to estimate the probability to reach a state which satisfies the property `expr`, within the time `bound`. The user must fix two main statistical parameters, α and ε, both in the real interval $]0,1[$. The answer provided by the tool is a confidence interval $[p-\varepsilon, p+\varepsilon]$ where α represents the probability of the answer of being wrong. The higher is the precision of the analysis the bigger must be the number of runs performed by the simulator. Thus the waiting time for a reply of a query depends both on

the length of runs, i.e. on the parameter bound, and on the statistical parameters ε and α.

In order to make feasible our analysis we try to understand if we can change some system parameters without affecting the quality of the analysis. In particular, we focus on the parameter C that is the number of slots composing a frame. The effectiveness of any synchronization protocol is crucially based on the exchange of some timing information to synchronize neighbor nodes. Said in other words, the longer nodes remain silent the quicker they get out of sync, because they do not get enough information to synchronize with each other. As consequence, once fixed the number of active slots, if the system gets out of sync with probability p, for a certain value of C, then the same system will get out of sync more quickly (or with higher probability) for a bigger value of C.

The parameter loss expresses the probability of message loss at the physical level due to the unreliability of the wireless medium. In our analysis, we will instantiate loss according to the results appeared in [17], where packet delivery performances of WSNs have been studied at physical layer under different transmission powers and physical-layer encodings. In that analysis, 60 Mica motes have been used to measure packet delivery under three different environmental settings: office building, open parking lot, habitat with moderate foliage. Under these settings, results show that the physical layer contributes to the packet-delivery performance, which is defined as the fraction of packets not successfully received by the receiver within a time window.

For the sake of simplicity, all nodes of our networks will be instantiated with the same value of the parameter loss. According to [17] this parameter will be set to: 10, to approximate the average message loss in a parking lot; 20, to model the average message loss in a office building; and 30, which represents the average message loss in an habitat setting with moderate foliage.

4.1 Verifying Clique Topologies

Paper [8] derives necessary and sufficient constraints on the guard times to guarantee the correctness of the protocol on clique topologies in the case of perfect communication. These constraints depend on the clock ratio min/max, on the parameter k_0 and on the maximal distance M between two transmitting slots[1]; they do not depend on the size N of the network. They are:

$$\begin{aligned} \mathsf{g} &> \left(1 - \tfrac{\mathsf{min}}{\mathsf{max}}\right) \cdot \mathsf{M} \cdot \mathsf{k}_0 + \tfrac{\mathsf{min}}{\mathsf{max}} \\ \mathsf{g} &< \left(1 - \tfrac{\mathsf{max}}{\mathsf{min}}\right) \cdot \mathsf{M} \cdot \mathsf{k}_0 + \mathsf{k}_0 - 2 \\ \mathsf{t} &> \left(1 - \tfrac{\mathsf{min}}{\mathsf{max}}\right) \cdot (\mathsf{k}_0 - \mathsf{g}) + \tfrac{\mathsf{min}}{\mathsf{max}} \end{aligned} \tag{2}$$

From an analysis of these conditions, paper [8] demonstrates that guard time values $\mathsf{g} = \mathsf{t} = 3$ are sufficient to guarantee clock synchronization in a clique of arbitrary size.

[1] For a formal definition of parameter M we refer to [8].

Table 2. On the parameter C

g	N	C	bound	frames	p	ε	α
3	10	12	$0.07 \cdot 10^9$	2	0.025	0.02	0.01
3	10	48	$0.28 \cdot 10^9$	2	0.029	0.02	0.01
3	10	192	$1.12 \cdot 10^9$	2	0.031	0.02	0.01
3	10	336	$2.00 \cdot 10^9$	2	0.031	0.02	0.01
4	15	17	$0.50 \cdot 10^9$	10	0.022	0.01	0.05
4	15	34	$1.00 \cdot 10^9$	10	0.027	0.01	0.05
4	15	68	$2.00 \cdot 10^9$	10	0.027	0.01	0.05

Here we want to demonstrate that, in the presence of packet loss, the size of the clique network *does* play a crucial role. In particular, the bigger is the clique the higher must be the value of g (and t) to ensure clock synchronization. Said in other words: *in the presence of message loss, fixed a value of* g *(and* t*), there is always a clique which gets out of sync with high probability.*

For networks with full connectivity clock synchronization means that all nodes of the network agree on the current slot. As a consequence, Definition 1 can be rephrased as in [8] in the following manner:

Definition 2. *A clique network is said to be synchronized if for all reachable states it holds the following:* $(\forall i, j \in \mathsf{Nodes})(\mathsf{SENDING}_i \implies \mathsf{csn}[i]{=}\mathsf{csn}[j])$.

So, in order to estimate the probability of going out of synchronization we will use UPPAAL SMC to perform the following quantitative check:

```
Pr[time<=bound] (<> exists(i:Nodes) exists(j:Nodes)
                    (WSN(i).SENDING and not(csn[i]==csn[j])))
```
(3)

Simulation setting In our simulations on cliques, all protocol parameters will satisfy the constraints in (2). As in [8], the guard time t is chosen to be the same as g. Parameter $\mathsf{tsn}[i]$ is chosen equal to i, as fully connectivity implies a different TX slot for each node. We set $\mathsf{k}_0 = 29$. Unfortunately, we cannot set $\mathsf{C} = 1129$, as in the real implementation, because the length of the runs that can be analyzed by UPPAAL is limited: in order to avoid integer overflow, the parameter bound cannot overtake the value $2 \cdot 10^9$. This means that if we would keep C close to the real value, then our execution runs would last for just a single time frame and they would be too short to provide any significant result. According to the discussion done in the preface of this section we perform our analysis for low values of the parameter C. This modification does not affect our analysis. As an example, in Table 2 we consider cliques with $\mathsf{N} = 10, 15$, $\mathsf{loss} = 20$ and $\mathsf{g} = 3, 4$. We then perform the quantitative check (3) by varying C and keeping constant the number of observed time frames. The value p represents the center of the confidence interval computed by UPPAAL SMC. Every check required 6623 runs of the protocol and lasted for about four days on a Intel core i5-2420M CPU 2.30GHz with 6G RAM. We gradually increased the precision of the parameter

Table 3. Cliques and node number. Maximal run length. $\alpha = 0.05$, $\varepsilon = 0.025$

N = 10 frames: 60			N = 15 frames: 40			N = 20 frames: 30			N = 30 frames: 20		
g	loss	p	g	loss	p	g	loss	p	g	loss	p
3	10	0.059	3	10	0.100	3	10	0.133	3	10	0.236
3	20	0.386	3	20	0.560	3	20	0.692	3	20	0.851
3	30	0.787	3	30	0.931	3	30	0.972	3	30	0.992
4	10	0.000	4	10	0.000	4	10	0.003	4	10	0.005
4	20	0.025	4	20	0.046	4	20	0.063	4	20	0.106
4	30	0.155	4	30	0.243	4	30	0.325	4	30	0.464

ε in order to achieve an interval which does not include the value 0 as a reply; in other words we have looked for lower bounds $p - \varepsilon > 0$.

Table 2 outlines that *when the number of time frames is fixed then the probability of going out of sync for the system does not decrease when the parameter* C *increases* (similar results can be obtained for different values of N, g, loss and for different topologies). As a consequence, our simulations provide a lower bound of the probability of getting out of sync in a setting with $\mathsf{C} = 1129$.

In Table 3 we study the behaviour of the protocol on cliques up to 30 nodes. We vary the number of nodes N, the guard time g and the parameter loss. Since we consider fully connected networks and the transmitting slots are grouped at the beginning of each time frame, we fix $\mathsf{C} = \mathsf{N}+2$, as in [8], to allow at least two idle slots at the end of each frame. We set the statistical parameters $\varepsilon = 0.025$, $\alpha = 0.05$ to have meaningful results. We check property (3) on the maximal run UPPAAL SMC can handle without incurring in integer overflow. The result of the quantitative check is represented by the probability p, which is the center of the confidence interval computed by UPPAAL SMC. Every check required 2952 runs of the protocol. In the following table we report the time required by our simulations on a Intel core i3-2310M CPU 2.10GHz with 4G RAM.

nodes	*time*
10	11 *hours*
15	1 *day* 7 *hours*
20	2 *days* 23 *hours*
30	9 *days* 4 *hours*

All runs in Table 3 are quite short (from 30 to 60 frames, depending on N); however, they are long enough to deduce some significant observation. For instance, we notice that once fixed the value of the guard time g, the probability of going out of sync increases when either N or loss increase. Moreover, once fixed both N and loss, the probability p decreases when the guard time g increases. Since the probability of going out of sync cannot decrease when going to longer runs, in Table 3 we compare probabilities associated to runs of different lengths. In particular, we notice that if we fix loss and g then the probability of getting out of sync increases when N increases. At the end of this section we will compare runs of the same length.

Table 4. Module comparison – $\mathsf{g} = 4$, $\alpha = 0.05$, run length 30 frames –

$\mathsf{N} = 5$			$\mathsf{N} = 10$		
$\mathcal{I}$	p	ε	$\mathcal{I}$	p	ε
$[0]$	0.019	0.01	$[0]$	0.015	0.01
$[0, \ldots, 4]$	0.113	0.030	$[0, \ldots, 4]$	0.113	0.030
$[0, \ldots, 9]$	0.665	0.050	$[0, \ldots, 9]$	0.677	0.030
$[0, \ldots, 14]$	0.827	0.050	$[0, \ldots, 14]$	0.816	0.030

The analysis provided in Table 3 says also that the protocol is certainly not suitable in certain scenarios. For instance, in a clique of at least 10 nodes with $\mathsf{g} = 3$ the system will get immediately out of sync with high probability if the loss probability is greater than 0.2. In other settings the results are not that strong. This is the case of a clique with 10 nodes, $\mathsf{g} = 4$ and $\mathsf{loss} = 20$. In this case, our analysis says that this system will get out of sync with probability 0.025. Such a value is ten times smaller than the loss probability, too small to conclude anything, at least in a so short run. Unfortunately, a priori, we cannot predict the behaviour of the system for longer runs as the probability p may increase or stabilize. In the following we will try to overcome this limitation.

UPPAAL SMC can simulate the behaviour of our systems on runs limited in size, called *execution modules*. At the beginning of an execution module all nodes are in the same time slot and with the same value in their clock variables. At the end of an execution module, UPPAAL SMC computes an estimate of the probability p to reach a state which does not satisfy Definition 2. This definition does not identify a single state of the system: nodes may have different clock values while still being in the same time slot. We claim that *the initial state, where all nodes begin the execution module with the same clock value, is the state which has the smallest probability to lead the system out of sync.* In order to support our argument, we provide an example in Table 4. We consider cliques of 5 and 10 nodes, with $\mathsf{g} = 4$, $\mathsf{C} = 7$ and $\mathsf{loss} = 20$. Table 4 shows experiments in which the system starts from a state that satisfies Definition 2 while internal clocks may have different values. The starting value of every internal clock is randomly chosen from a fixed interval $\mathcal{I}$ of clock values. Runs are 30 time frames long. We set $\alpha = 0.05$. It can be noticed that the smallest desync probability is obtained when the execution module starts in the initial state where all nodes have the same clock value. Similar results can be obtained for other values of N, g, C and loss.

In virtue of this observation, we can divide a long run in consecutive execution modules, all starting in the initial state. Then, we can derive by composition a lower bound of the probability of desynchronization for that run. Thus, if $[p - \varepsilon, p + \varepsilon]$ is the confidence interval provided by UPPAAL SMC after performing the quantitative check (3) within an execution module, then the probability of going out of sync within n execution modules is *at least*

$$1 - (1 - (p - \varepsilon))^n. \tag{4}$$

Table 5. Quantitative check on cliques with $\mathsf{loss} = 20$ and $\alpha = 0.05$

g	N	C	$p - \varepsilon$	300 frames	600 frames	900 frames
3	10	12	0.361	≥ 0.893	≥ 0.989	≥ 0.999
3	15	17	0.535	≥ 0.998	≥ 0.999	≥ 0.999
3	20	22	0.667	≥ 0.999	≥ 0.999	≥ 0.999
3	30	32	0.826	≥ 0.999	≥ 0.999	≥ 0.999
4	15	17	0.021	≥ 0.156	≥ 0.273	≥ 0.373
4	20	22	0.038	≥ 0.321	≥ 0.539	≥ 0.687
4	30	32	0.081	≥ 0.718	≥ 0.920	≥ 0.980

Table 5 extends the results of Table 3 to longer runs which lasts for 300, 600 and 900 time frames, respectively. The forth column of Table 5 reports the lower bound of the confidence interval of an execution module. When $\mathsf{N} = 10, 15, 20, 30$ the execution module studied in Table 3 lasts for approximately $60, 40, 30$ and 20 time frames respectively. Thus, by applying the formula (4) with $n = 5, 8, 10, 15$ we obtain a lower bound for the probability of being out of sync within 300 time frames in the cases of cliques with 10, 15, 20 and 30 nodes, respectively. Analogously when $n = 10, 15, 20, 30$ and $n = 15, 22, 30, 45$ we obtain a lower bound for the probability of being out of sync within 600 and 900 time frames, respectively.

As discussed at the beginning of this section, the values of Table 5 represent also lower bounds of the probability of desynchronization for the real implementation. In the real setting, with $\mathsf{C} = 1129$ and clock frequency of 32 kHz, a time frame lasts for about $1sec$. As a consequence, Table 5 expresses a lower bound of the probability of getting out of sync within 5, 10 and 15 *min*. Thus, when $\mathsf{g} = 3$ the probability of getting out of sync is high also for small networks enough for small networks (around 10 nodes), but when $\mathsf{N} = 15$ we have a probability of being out of sync of almost 0.4 in $15min$. When $\mathsf{N} = 20$ the probability reaches 0.7 in less than $15min$. When $\mathsf{N} = 30$ the probability reaches 0.7 in less than $5min$. These results outline an increasing of the desync probability when the number of nodes increases.

4.2 Verifying Grid Topologies

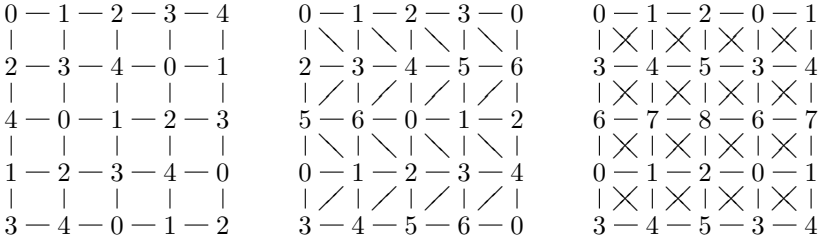
Clock synchronization in clique topologies has been studied in [8] as a first step towards more realistic topologies. Usually sensor nodes have a limited number of neighbours and do not have direct communication with the whole network. In this section, we study how the gMAC synchronization protocol behaves on regular topologies which better simulate a uniform node distribution in a given area[2]. In particular, we will focus on grid topologies where nodes have a uniform number of neighbours. Unlike cliques, there are no theoretical results suggesting how to choose protocol parameters to guarantee the synchronization of grid

[2] Regular topologies have been applied to WSNs to study coverage, connectivity and energy-efficiency.

networks in the case of non-lossy communication. The implementation of gMAC adopts an high guard time, $\mathsf{g} = 9$, to ensure synchronization in networks with arbitrary topologies. In this section, we study whether high values of g guarantee node synchronization in grid-based networks, in the case of lossy communication.

In our simulations we focus on a small sensor network where nodes are placed in a 5×5 grid, thus $\mathsf{N} = 25$. Unlike cliques, grid topologies do not need a different TX slot for each node: we can allocate the same TX slot to different nodes provided that when two nodes are neighbours or have a common neighbor then they get distinct TX slot numbers. According to the implementation of gMAC, where the number of TX slots is limited, we consider the minimum number of TX slots to be allocated to satisfy conditions (1). The number of transmission slots depends on the number of neighbours for each single node; if a node v has k neighbours then we need a TX slot in which v transmits and all its neighbours listen, and k distinct slots in which each neighbor transmits and v listens. Thus if k represents the maximum node degree, then we need at least $k + 1$ TX slots.

In the following we analyze the behaviour of the protocol on grid topologies by considering three possible maximum node degrees: 4, 6 and 8. These grid networks require at least 5, 7 and 9 TX slots, respectively. Below we report the three topologies we consider along with a simple slot allocation which satisfies conditions (1) by using exactly $k + 1$ TX slots, where k is the maximum node degree. The grid structures outlines the network topology while the identifiers $0, 1, \ldots$ show the TX slot allocated for the corresponding node.



As for cliques, our analysis does not loose in generality if we consider small values of C. Thus we pick $\mathsf{C} = 7, 9, 11$ for the three different cases, respectively. Depending on the maximum node degree, a single time frame is composed by $5, 7, 9$ TX slots plus 2 idle slots. TX slots are allocated according to the distributions depicted above. We set $\mathsf{k}_0 = 29$ and we vary the parameter loss, as done for cliques. Then, we apply UPPAAL SMC to perform the following quantitative check, according to Definition 1:

```
Pr[time<=bound](<> exists(i:Nodes) exists(j:Nodes)
  (neighbor(i,j) and WSN(i).SENDING and not(csn[i]==csn[j]))
```
(5)

Again, we consider the longest run UPPAAL SMC can handle to avoid integer overflow by setting $\texttt{bound} = 2 \cdot 10^9$. This means that an execution module lasts for almost 100, 80 and 65 time frames when the maximum node degree is 4, 6 and 8, respectively. These are quite short runs, but long enough to conclude that

Table 6. Grids 5×5 and node degree. Maximal run length. $\alpha = 0.05$, $\varepsilon = 0.03$

max degree 4			
g	C	loss	p
6	7	0	0
6	7	10	0.03
6	7	20	0.11
6	7	30	0.45
7	7	0	0
7	7	10	0.01
7	7	20	0.06
7	7	30	0.28

max degree 6			
g	C	loss	p
6	9	0	0
6	9	10	0.01
6	9	20	0.07
6	9	30	0.25
7	9	0	0
7	9	10	0
7	9	20	0.03
7	9	30	0.18

max degree 8			
g	C	loss	p
6	11	0	0
6	11	10	0
6	11	20	0.05
6	11	30	0.19
7	11	0	0
7	11	10	0
7	11	20	0.02
7	11	30	0.11

Table 7. Quantitative check on 5×5 grids with loss $= 20$ and g $= 6$

degree	C	$p - \varepsilon$	900 frames	1800 frames	2700 frames	3600 frames
4	7	0.08	≥ 0.53	≥ 0.78	≥ 0.90	≥ 0.95
6	9	0.04	≥ 0.39	≥ 0.61	≥ 0.76	≥ 0.84
8	11	0.02	≥ 0.25	≥ 0.44	≥ 0.58	≥ 0.69

the system may get out of sync also for high values of the guard time g. The result of the quantitative check is reported in Table 6. The value p represents the center of the confidence interval computed by UPPAAL.

The compositional reasoning on execution modules adopted for cliques can be easily generalized to grid topologies. Table 7 fixes loss $= 20$ and g $= 6$. It reports lower bounds to the probability of getting out of sync within 900, 1800, 2700 and 3600 time frames. As said before, in the real implementation a time frame lasts for around $1sec$. Thus, when considering g $= 6$ and a message loss of 20%, we observe that the desync probability exceeds 0.5 in less than $15min$ for degree 4, in less than $30min$ for degree 6, and in less than $45min$ for degree 8. Table 7 outlines also how the performances of the protocol depend on the node degree: the probability of getting out of sync decreases for grid topologies with higher node degree.

Finally, let us give a taste of what happens when g $= 7$. Among the results on Table 6 we extend the case of degree 4 and loss $= 20$, where the probability of getting out of sync within 100 time frames lays in the interval $[0.03\,,\,0.09]$. The projection to 2700 time frames says that the probability of getting out of sync becomes greater than 0.54. In the real settings, this means that the probability of getting out of sync exceeds 0.5 in less than $45min$.

In conclusion, in the case of lossy communication, *small grid topologies have a high probability of getting out of sync even for high values of the guard time* g *and for low values of the loss probability. Moreover the probability of getting out of sync increases when decreasing the maximum node degree.*

5 Conclusions, Future and Related Work

Our work has been strongly inspired by a recent analysis [8] of the gMAC synchronization protocol on clique and line topologies, in the ideal case of non-lossy communication. That analysis provides constraints on the protocol parameters that are both necessary and sufficient for the correctness of the protocol for cliques of arbitrary size. Here we have carried on the work of [8] in the case of lossy communication. We have extended their model and obtained a network of probabilistic timed automata [6] which has been used for doing Statistical Model Checking within the UPPAAL toolset [7]. As a main result, we have showed that in the presence of message loss the constraints provided in [8] may be not sufficient to ensure clock synchronization of cliques of arbitrary size. Then, we have extended our analysis of the protocol to small grid topologies and again found that, in the case of lossy communication, the nodes of the grid may get out of sync with high probability. More interestingly, grid topologies with higher node degree have a smaller probability of desynchronization. This lets us to conjecture that higher connectivity helps synchronization protocols. In this respect, among the regular topologies, clique topologies are those with the best performances!

As in [8] we have assumed a fixed slot allocation. However, the implementation of gMAC includes a probabilistic dynamic slot allocation algorithm. The only analysis we are aware of the probabilistic gMAC algorithm appears in [16]. In that paper, mobile sensors do not use a fixed schedule to control medium access but instead employ gMAC's full decentralized slot allocation: gossiping is introduced to allow each node to decide when to send. Paper [16] analyzes the energy-efficiency of gMAC under the assumption of perfect clock synchronization. The protocol, formalised in the MoDeST language [4], is evaluated using the discrete-event simulator of the Möbius tool suite. We are planning to study the performance of the gMAC protocol with dynamic slot allocation in the case of lossy communication and realistic clock. In doing that, we intend to adopt either a (truncated) normal distribution or a (truncated) exponential distribution for modeling a more realistic delay between consecutive ticks.

Statistical Model Checking allows us to study networks of bigger size with respect to the state-of-the art model checking technology, such as PRISM [9,11]. SMC can be seen as a trade off between testing and formal verification: its approach consists in performing an appropriate number of simulations which are elaborated with statistical algorithms to verify if a given property is satisfied with a certain probability. Unlike an exhaustive approach, a simulation-based solution does not guarantee a correct result with a 100% confidence. It is only possible to bound the probability of making an error. In order to study bigger systems with an higher confidence, paper [5] proposes a distributed implementation of UPPAAL SMC by means of a master/slave architecture where several computers are used to generate simulations and a single master process is used to collect those simulations and perform the statistical test. We are planning to employ this approach to extend the confidence of the results we obtained in this paper.

Acknowledgments. The anonymous referees provided insightful comments.

References

1. Assegei, F.: Decentralized frame synchronization of a TDMA-based wireless sensor network. Master's thesis, Eindhoven University of Technology, Department of Electrical Engineering (2008)
2. Behrmann, G., David, A., Larsen, K.G.: c. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer (2004)
3. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: QEST 2006. pp. 125–126. IEEE Computer Society (2006)
4. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MODEST: A compositional modeling formalism for hard and softly timed systems. IEEE Trans. Software Eng. 32(10), 812–830 (2006)
5. Bulychev, P.E., David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Checking and distributing statistical model checking. In: Goodloe, A., Person, S. (eds.) NFM 2012. LNCS, vol. 7226. Springer (2012)
6. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 80–96. Springer (2011)
7. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer (2011)
8. Heidarian, F., Schmaltz, J., Vaandrager, F.W.: Analysis of a clock synchronization protocol for wireless sensor networks. Theor. Comput. Sci. 413(1), 87–105 (2012)
9. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer (2011)
10. Nordic Semiconductors: nRF2401 Single-chip 2.4GHz Transceiver Data Sheet (2002)
11. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Formal Methods in System Design (2012), to appear
12. QUASIMODO: Preliminary description of case studies, deliverable 5.2 from the FP7 ICT STREP project 214755 (January 2009)
13. Sundararaman, B., Buy, U., Kshemkalyani, A.D.: Clock synchronization for wireless sensor networks: a survey. Ad Hoc Networks 3(3), 281–323 (2005)
14. van Vessem, I.: WSN gMAC protocol specifications. Tech. rep., CHESS B.V., Haarlem, NL (2008), version 1.1. Patent pending US 12 / 250,040
15. Wu, Y.C., Chaudhari, Q.M., Serpedin, E.: Clock synchronization of wireless sensor networks. IEEE Signal Process. Mag. 28(1), 124–138 (2011)
16. Yue, H., Bohnenkamp, H.C., Katoen, J.P.: Analyzing energy consumption in a gossiping mac protocol. In: Müller-Clostermann, B., Echtle, K., Rathgeb, E.P. (eds.) MMB&DFT 2010. LNCS, vol. 5987, pp. 107–119. Springer (2010)
17. Zhao, J., Govindan, R.: Understanding packet delivery performance in dense wireless sensor networks. In: Akyildiz, I.F., Estrin, D., Culler, D.E., Srivastava, M.B. (eds.) SenSys 2003. pp. 1–13. ACM (2003)