Analysis of a Clock Synchronization Protocol for Wireless Sensor Networks☆

Faranak Heidarian[a,*], Julien Schmaltz[a,b], Frits Vaandrager[a]

[a] *Institute for Computing and Information Sciences*
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
[b] *School of Computer Science*
Open University of the Netherlands
P.O. Box 2960, 6401 DL Heerlen, The Netherlands

Abstract

The Dutch company Chess develops a wireless sensor network (WSN) platform using an epidemic communication model. One of the greatest challenges in the design is to find suitable mechanisms for clock synchronization. In this paper, we study a proposed clock synchronization protocol for the Chess platform. First, we model the protocol as a network of timed automata and verify various instances using the Uppaal model checker. Next, we present a full parametric analysis of the protocol for the special case of cliques (networks with full connectivity), that is, we give constraints on the parameters that are both necessary and sufficient for correctness. These results have been checked using the proof assistant Isabelle. We report on the exhaustive analysis of the protocol for networks with four nodes, and we present a negative result for the special case of line topologies: for any instantiation of the parameters, the protocol will eventually fail if the network grows.

Keywords: industrial application, clock synchronization, timed automata, model checking, theorem proving, wireless sensor networks

1. Introduction

Wireless sensor networks (WSNs) consist of potentially thousands of autonomous devices that communicate via radio and use sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound or motion, at different locations. WSNs have numerous applications, ranging from monitoring of dikes to smart kindergartens, and from forest fire detection to monitoring of the Matterhorn.

The Dutch company Chess develops a WSN platform using an epidemic (gossip) communication model [3]. Gossiping in distributed systems refers to the repeated probabilistic exchange of information between two members [4, 5]. The effect is that information can spread within a group just as it would in real life. Their simplicity, robustness and flexibility make gossip based algorithms attractive for data dissemination and aggregation in wireless sensor networks. However, formal analysis of gossip algorithms is a challenging research problem [6]. The Chess WSN currently distinguishes three protocol layers: the *Medium Access Control (MAC) layer*, which is responsible for regulating the access to the wireless shared channel, the intermediate *Gossip layer*, which is responsible for insertion of new messages, forwarding of current messages and

☆Research supported by the European Community's Seventh Framework Programme under grant agreement no 214755 (QUASIMODO). An extended abstract of this report appeared as [1]. A preliminary version of the model presented in this paper appeared in [2].

*Research supported by NWO/EW project 612.064.610 Abstraction Refinement for Timed Systems (ARTS).

Email addresses: `F.Heidarian@cs.ru.nl` (Faranak Heidarian), `Julien.Schmaltz@ou.nl` (Julien Schmaltz), `F.Vaandrager@cs.ru.nl` (Frits Vaandrager)

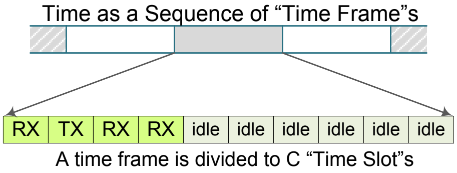
Time as a Sequence of "Time Frame"s
RX
TX
RX
RX
idle
idle
idle
idle
idle
idle
A time frame is divided to C "Time Slot"s

Figure 1: The structure of a time frame

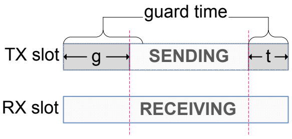

Figure 2: TX and RX slots

deletion of old messages, and the *Application layer*, which has the business logic that interprets messages and may generate new messages. In our research we focus on the MAC layer of the Chess WSN. Characteristics of the other layers influence the design decisions for the MAC layer. For instance, the redundant nature of the Gossip layer justifies occasional message loss in the MAC layer.

The MAC layer uses a Time Division Multiple Access (TDMA) protocol. Time is divided in fixed length *frames*, and each frame is subdivided into *slots* (see Figure 1). Slots can be either *active* or *idle*. During active slots, a node is either listening for incoming messages from neighboring nodes ("RX") or it is sending a message itself ("TX"). During idle slots a node is switched to energy saving mode. Nodes are battery operated devices with an expected uninterrupted field deployment of several years. Hence, energy efficiency is a major concern in the design of WSNs. For this reason, the number of active slots is typically much smaller than the total number of slots (less than 1% in the current implementation [3]). The active slots are placed in one contiguous sequence which currently is placed at the beginning of the frame. A node can only transmit a message once per time frame in its TX slot. If two neighboring nodes choose the same send slot, a collision will occur in the intersection of their ranges preventing message delivery of either node's message in that intersection. Ideally, no neighboring pair would ever choose the same send slot. This has proven to be very hard to achieve, especially in settings with node mobility. In our work, we have not addressed the issue of slot allocation and simply assume that the TX slots of all nodes are fixed and have been chosen in such a way that no collisions occur.

One of the greatest challenges in the design of the MAC layer is to find suitable mechanisms for clock synchronization: we must ensure that whenever some node is sending all its neighbors are listening. In this paper, we study a proposed clock synchronization algorithm for the Chess platform. Each wireless sensor node comes equipped with a low-cost 32 KHz crystal oscillator that drives an internal clock that is used to determine the start and end of each slot. This may cause the TDMA time slot boundaries to drift and thus lead to situations in which nodes get out of sync. To overcome this problem, the notion of *guard time* is introduced: at the beginning of its TX slot, before actually starting transmission, a sender waits a certain amount of time for the receiver to be ready to receive messages. Similarly, the sender also waits for some time period at the end of its TX slot (see Figure 2). In the current implementation, each slot consists of 29 clock cycles, out of which 18 cycles are used as guard time. Assegei [7] calculated how the battery life of a wireless sensor node is influenced by the guard time. Figure 3, taken from [7], summarizes these results.

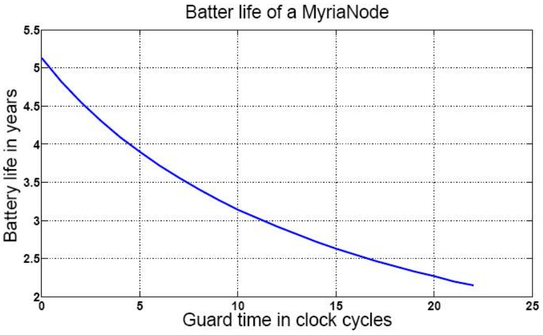

Figure 3: Battery life as a function of guard time

Clearly, it is of vital importance to reduce the guard time as much as possible, since this directly affects the battery life, which is a key characteristics of WSNs. Reduction of the guard time is possible if the hardware clocks are properly synchronized.

Many clock synchronization protocols have been proposed for WSNs. In most of these protocols, clocks are synchronized to an accurate real-time standard like Coordinated Universal Time (UTC). We refer to [8] for an overview of this type of protocols. However, these protocols are based on the exchange of time stamp messages, and for the Chess WSN this creates an unacceptable computation and communication overhead. It is possible to come up with more efficient algorithms, since for the MAC layer a weak form of clock synchronization suffices: a node only needs to be synchronized to its immediate neighbors, not to faraway nodes or to UTC. Fan & Lynch [9] study the *gradient clock synchronization (GCS)* problem, in which the difference between any two network nodes' clocks must be bounded from above by a non-decreasing function. Thus nearby nodes must be closely synchronized but faraway nodes are allowed to be more loosely synchronized. In the approach of Fan & Lynch [9], nodes compute logical clock values based on their hardware clocks and message exchanges, and the goal is to synchronize the nodes' logical clocks as closely as possible, while satisfying certain validity conditions. Logical clocks have been introduced by Lamport [10] to totally order the events in a distributed system. A key property of Lamport's logical clocks is that they never run backwards: their value can only increase. In fact, Fan & Lynch [9] assume a constant lower bound on clock speed. Also Meier & Thiele [11] and Pussente & Barbosa [12], who adapt the work of Fan & Lynch to the setting of wireless sensor networks, make a similar assumption (with minimal clock rates $\frac{1}{2}$ and $\frac{1}{\mathsf{D}}$, respectively, where D is the network diameter). For certain applications of WSNs it is important to have Lamport style logical clocks. For example, if two sensor nodes observe a moving object, then logical clocks allow one to establish the object's direction by determining which node observed the object first [11]. However, for the MAC layer there is no need to compute a total order on events: we only need to ensure that whenever one node is sending all neighbors are listening. Since it is allowed to set back clocks, the lower bounds of [9, 11] do not apply in this case.

Meier & Thiele [11] provide a lower bound for the achievable synchronization quality in sensor networks, but no algorithms that attain or come close to this bound. Pussente & Barbosa [12] do present an algorithm, but this cannot be applied in the TDMA based setting of the Chess algorithm. Basic assumptions of [11, 12] are that (a) messages sent between neighbors are always delivered instantaneously, and (b) consecutive communications between any two neighbors in the same direction are no farther apart in time than some given time d. Pussente & Barbosa [12] derive a strict upper bound of $c+2(1+2\hat{\rho})d$ on the difference between the clocks of neighboring nodes, where $c > 0$ is a constant and $\hat{\rho} \in [0, 1)$ is the maximal clock drift. But since this bound exceeds $2d$ and in a TDMA setting d basically equals the length of a frame, the algorithm of [12] is unable to guarantee that whenever some node is sending all its neighbors are listening.

The current implementation of the Chess WSN uses *Median*, an extension of an algorithm proposed by Tjoa et al [13]. The idea is that in every frame each node computes its phase error to any of its direct neighbors. After the last active slot, each node adjusts its clock by the median of the phase errors of its immediate neighbors. Assegei [7] points out that the performance of the Median algorithm decreases if the network becomes more dynamic. In fact, in [14] we established — using Uppaal — that in certain cases even a static, fully synchronized network may become unsynchronized if the Median algorithm is used, even in a setting with infinitesimal clock drifts. The theoretical result was later reproduced experimentally in a real network of Chess. Assegei [7] proposes a variation of the Median algorithm that uses Kalman filters, but in [15] we show that also this variations leads to serious synchronization problems. In this paper, we use formal methods to analyze another variation of the Chess algorithm in which a node adjusts its clock whenever a message arrives. Advantages of this approach are (a) unlike the Median approach and its variants we need almost no guard time at the end of a sending slot (2 clock ticks suffice instead of 9 ticks in the current implementation), and (b) the computational overhead becomes essentially zero. However, the practical usefulness of our algorithm still needs to be explored further.

The research reported in this paper was carried out within the context of the EU project Quasimodo. The main goal of Quasimodo was to develop new techniques and tools for model-driven design, analysis, testing and code-generation for advanced embedded systems where ensuring quantitative bounds on resource consumption is a central problem. Case studies have been the driving momentum behind the project. Quasimodo followed an iterative approach where fundamental research on theory and algorithms —challenged by real-life case studies— was developed and implemented in methods and tools, which were evaluated through case studies. Quasimodo research on the Chess WSN case study had a significant impact on the design of the network [16]. But equally important, it also provided major challenges for research on theory and algorithms for model checking. Most industrial applications of Uppaal thus far involve small networks with a fixed topology. The size and dynamic nature of the Chess WSN, and the resulting complexity of clock synchronization, provides a major challenge for model checking technology that goes beyond what we have seen in other Uppaal case studies. Another challenge raised by the present paper is in the area of parametric model checking: the parameter constraints that we derive in Section 4 are nonlinear, and appear to be beyond reach of existing algorithms for parametric model checking.

The rest of this paper is structured as follows. In Section 2, we model the algorithm using timed automata. Section 3 describes the use of the timed automata model checker UPPAAL [17, 18] to analyze WSNs with full connectivity. We verify various instances and identify three different scenarios that may lead to situations where the network is out of sync. Section 4 presents a full parametric analysis of the protocol for cliques (networks with a connection between every pair of nodes), that is, we give constraints on the parameters that are both necessary and sufficient for correctness. We have checked our results using the proof assistant Isabelle [19]. In Section 5 we report on an exhaustive analysis using Uppaal of all networks with 4 nodes. Section 6 presents a result for the special case of line topologies: for any instantiation of the parameters, the protocol will eventually fail if the network grows. Section 7, finally, discusses related work and draws conclusions.

UPPAAL models, Isabelle sources and invariant proofs for this paper are available at
`http://www.mbsd.cs.ru.nl/publications/papers/fvaan/HSV09/`.

Acknowledgement. Many thanks to Frits van der Wateren, Marcel Verhoef and Bert Bos from Chess for explaining their WSN algorithms to us. Thanks also to the anonymous reviewers for suggestions that helped us to improve our paper.

2. Uppaal Model

In this section, we describe the UPPAAL model that we constructed of the Chess protocol. For a detailed account of the timed automata model checking tool UPPAAL, we refer to [17, 18] and to the website `http://www.uppaal.com`.

We assume a finite, fixed set of wireless nodes $\mathsf{Nodes} = \{0, \ldots, \mathsf{N} - 1\}$. The behavior of each individual node $i \in \mathsf{Nodes}$ is described by three timed automata: **Clock**(i), **WSN**(i) and **Synchronizer**(i). Automaton **Clock**(i) models the hardware clock of the node, automaton **WSN**(i) takes care of sending messages, and the **Synchronizer**(i) automaton resynchronizes the hardware clock upon receipt of a message. The complete model consists of the composition of timed automata **Clock**(i), **WSN**(i) and **Synchronizer**(i), for each $i \in \mathsf{Nodes}$.

Figure 4 illustrates the architecture of our model for a single node i. For each i, there is a state variable $\mathsf{clk}[i]$ that records the (integer) value of i's hardware clock (initially 0), and a variable $\mathsf{csn}[i]$ that records the current slot number of node i (also 0 initially). Variable $\mathsf{clk}[i]$ is incremented cyclically whenever **Clock**[i] ticks, but it can also be reset by **Synchronizer**[i]. Automaton **WSN**[i] reads $\mathsf{clk}[i]$ in order to determine when to transmit. Automaton **WSN**[i] both reads and write variable $\mathsf{csn}[i]$. The **Synchronizer**[i] needs to read variable $\mathsf{csn}[i]$ in order to determine whether node i is active or idle. In UPPAAL, a broadcast channel can be used to synchronize transitions of multiple automata. If a is a broadcast channel and one automaton in the network is in a state with an outgoing $a!$ transition, then this transition may always occur (provided the guard evaluates to true). In this case, the transition synchronizes with the $a?$ transitions of all automata that enable such a transition. Automata that do not enable an $a?$ transition remain in the same state. Within our model, broadcast channel $\mathsf{tick}[i]$ is used to synchronize the activities within node i, and broadcast channel $\mathsf{start\_message}[i]$ is used to inform all the nodes in the network that node i has started transmission. More specifically, automaton **WSN**[i] performs a $\mathsf{start\_message}[i]!$ action to indicate

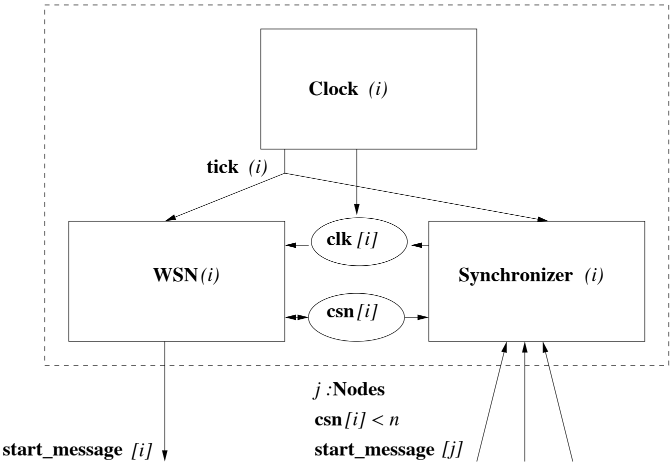

Figure 4: Model architecture for node i

that node i starts transmission, and whenever some node j starts transmission and node i is in an active slot ($\mathsf{csn}[i] < n$), automaton **Synchronizer**[i] may perform a $\mathsf{start\_message}[j]?$ transition.

Table 1 lists the parameters (constants in UPPAAL terminology) that we use in our model, together with some basic constraints. The domain of all parameters is the set of natural numbers.

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	$0 < \mathsf{n} \leq \mathsf{C}$
$\mathsf{tsn}[i]$	TX slot number for node $i \in \mathsf{Nodes}$	$0 \leq \mathsf{tsn}[i] < \mathsf{n}$
k_0	number of clock ticks in a time slot	$0 < \mathsf{k}_0$
g	guard time	$0 < \mathsf{g}$
t	tail time	$0 < \mathsf{t}$, $\mathsf{g} + \mathsf{t} + 2 \leq \mathsf{k}_0$
min	minimal time between two clock ticks	$0 < \mathsf{min}$
max	maximal time between two clock ticks	$\mathsf{min} \leq \mathsf{max}$

Table 1: Protocol parameters

2.1. Clock

Timed automaton **Clock**(i), displayed in Figure 5, models the behavior of the hardware clock of node i. It has a single location and a single transition. It comes equipped with a local clock variable x, which is initially 0, that is used to measure the time in between clock ticks. Whenever x reaches the value min, the automaton enables a $\mathsf{tick}[i]!$ action. The $\mathsf{tick}[i]!$ action must occur before x has reached value max. Then x is reset to 0 and the (integer) value of i's hardware clock $\mathsf{clk}[i]$ is incremented by 1. For convenience and in order to make model checking feasible, we reset the hardware clock after k_0 ticks, that is, the clock takes integer values modulo k_0 (we use UPPAAL's modulo operator %). This is not an essential modeling assumption and we can easily change this. In the implementation of the protocol, the clock domain is larger and additional program variables are used to record, for instance, the clock value at which a frame starts. However, in order to avoid state space explosion, we try to reduce the number of state variables and the domains of these variables.

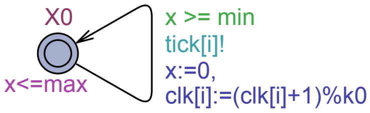

Figure 5: Timed automaton **Clock**(i)

2.2. *Wireless Sensor Node*

Automaton **WSN**(i), displayed in Figure 6, is the most important component in our model. It has three locations and four transitions. The automaton stays in initial location WAIT until the current slot number of i equals the TX slot number of i ($\mathsf{csn}[i] = \mathsf{tsn}[i]$) and the g^{th} clock tick in this slot occurs. It then jumps to location GO_SEND. This is an urgent location that is left immediately via a start_message[i]!-transition to location SENDING. Broadcast channel start_message[i] is used to inform all neigboring nodes that a new message transmission has started. The automaton stays in location SENDING until the start of the tail interval, that is, until the $(\mathsf{k}_0 - \mathsf{t})^{th}$ tick in the current slot (cf. Figure 2), and then returns to location WAIT. At the end of each slot, that is, when the $\mathsf{k}_0{}^{th}$ tick occurs, the automaton increments its current slot number (modulo C).

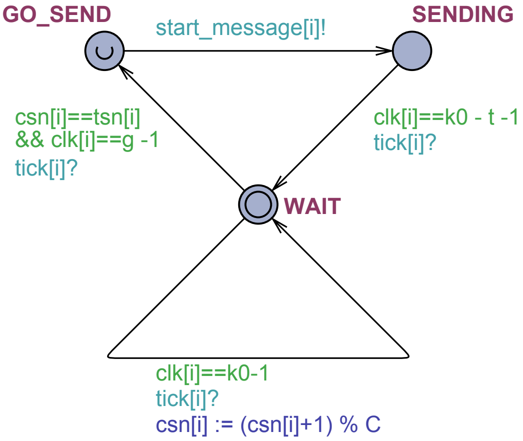

Figure 6: Timed automaton **WSN**(i)

2.3. *Synchronizer*

Automaton **Synchronizer**(i), displayed in Figure 7, is the last component of our model. It performs the role of the clock synchronizer in the TDMA protocol. The automaton has two locations and two transitions. The automaton waits in its initial location S0 until it detects the start of a new message, that is, until a start_message[j]? event occurs, for some j. We use the Uppaal select statement to nondeterministically select a $j \in \mathsf{Nodes}$. The automaton then moves to location S1, provided node i is active ($\mathsf{csn}[i] < \mathsf{n}$). Remember that at the moment when the start_message[j]? event occurs, the hardware clock of node j, $\mathsf{clk}[j]$, has value g. Therefore, node i resets its own hardware clock $\mathsf{clk}[i]$ to $\mathsf{g} + 1$ upon occurrence of the first clock tick following the start_message[j]? event. The automaton then returns to its initial location S0.

In our model there is no delay between sending and receipt of messages. Following Meier & Thiele [11], we assume delay uncertainties to be negligible, and we therefore eliminate the delays themselves from our

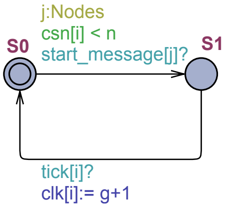

Figure 7: Timed automaton **Synchronizer**(i)

analysis. When communication is infrequent, this is reasonable since the impact of clock drift dominates over the influence of delay uncertainties.

Automaton **Synchronizer**(i) has no constraint on the value of j, that is, we assume that node i can receive messages from any node in the network. Hence the network has full connectivity. It is easy to generalize our model to a setting with arbitrary network topologies by adding a guard $\mathsf{neighbor}(i, j)$ to the transition from S0 to S1 that indicates that i is a direct neighbor of j. We assume $\mathsf{neighbor}(i, j) \Rightarrow i \neq j$. The $\mathsf{neighbor}(i, j)$ predicate does not have to be symmetric since in a wireless sensor network it may occur that i can receive messages from j, but not vice versa. For networks with full connectivity, we assume that all nodes have unique TX slot numbers:

$$i \neq j \quad \Rightarrow \quad \mathsf{tsn}[i] \neq \mathsf{tsn}[j] \tag{1}$$

For networks that are not fully connected, this assumption generalizes to the requirements that neighboring nodes have distinct TX slot numbers, and distinct nodes with the same TX slot number do not have a common neighbor:

$$\mathsf{neighbor}(i, j) \quad \Rightarrow \quad \mathsf{tsn}[i] \neq \mathsf{tsn}[j] \tag{2}$$

$$\mathsf{neighbor}(i, j) \wedge \mathsf{neighbor}(i, k) \quad \Rightarrow \quad \mathsf{tsn}[j] \neq \mathsf{tsn}[k] \tag{3}$$

3. Uppaal Analysis Results for Cliques

We call a wireless sensor network *synchronized* if, whenever a node is sending, all neighboring nodes have the same slot number as the sending node. For networks with full connectivity this means that all nodes in the network agree on the current slot. We obtain the following formal definition of correctness.

Definition 1. A network with full connectivity is *synchronized* if and only if for all reachable states $(\forall i, j \in \mathsf{Nodes})(\mathsf{SENDING}_i \;\Rightarrow\; \mathsf{csn}[i] = \mathsf{csn}[j])$.

Our objective is to find necessary and sufficient constraints on the system parameters that ensure that a network with full connectivity is synchronized. To this end, we assigned different values to the parameters of the model and used Uppaal to verify the property of Definition 1. Based on the outcomes (and in particular the counterexamples generated by Uppaal) we derived general constraints. For networks with up to 4 nodes, the Uppaal model checker is able to explore the state space within a few seconds.

Table 2 shows some example values of the parameters for which the model is synchronized. In fact, via a series of model checking experiments, using binary search, we found that the values of min and max in this table are the smallest consecutive natural numbers for which the model with the values assigned to N, C, n, k_0 and g is synchronized. Note that for correctness of the WSN algorithm the exact values of min and max are not important: what matters is their ratio. By setting $\mathsf{min} = m$, $\mathsf{max} = m + 1$ and letting m grow, the hardware becomes more and more accurate, until (hopefully) we reach the point at which the algorithm becomes correct. Parameter t is chosen equal to g and $\mathsf{tsn}(i)$ is chosen equal to i. We keep n, k_0 and g constant and vary C, the number of slots in a frame. Observe that if the value of C increases also the values of min and max increase, i.e., if the length of a frame increases then the hardware clocks must become more accurate to maintain synchronization.

N	2			3			4		
C	6	8	10	6	8	10	6	8	10
n	4	4	4	4	4	4	4	4	4
k_0	10	10	10	10	10	10	10	10	10
g	2	2	2	2	2	2	2	2	2
min	49	69	89	39	59	79	29	49	69
max	50	70	90	40	60	80	30	50	70

Table 2: Some Uppaal verification results

Observe that these parameter values are not realistic: a realistic clock accuracy is around 30 ppm (parts-per-million), C is about 1000 (instead of 10), and g is 9 (instead of 2). Uppaal cannot handle realistic values because of the state explosion problem. Nevertheless, as we will see, the counterexamples provided by Uppaal do provide insight.

In Table 3, we keep all the parameters constant and then consider the values of min and max for different numbers of nodes when n changes. Since, in accordance with the specification of the protocol, we only consider slot allocations in which the sending slots are placed at the very beginning of a frame, increasing n has no impact on network behavior: when no node is transmitting anyway, it makes no difference whether nodes are sleeping or listening. In Table 4, we keep all the parameters constant and then consider the smallest values of min and max for different number of nodes when k_0 changes. It turns out that increasing k_0 forces us to increase min and max. In Table 5, we keep all the parameters constant and then consider the smallest values of min and max for different number of nodes when g changes. Increasing g, allows us to to decrease min and max.

N	2				3				4			
C	20	20	20	20	20	20	20	20	20	20	20	20
n	4	5	10	15	4	5	10	15	4	5	10	15
k_0	10	10	10	10	10	10	10	10	10	10	10	10
g	2	2	2	2	2	2	2	2	2	2	2	2
min	189	189	189	189	179	179	179	179	169	169	169	169
max	190	190	190	190	180	180	180	180	170	170	170	170

Table 3: Numerical results, changing n

N	2				3				4			
C	6	6	6	6	6	6	6	6	6	6	6	6
n	4	4	4	4	4	4	4	4	4	4	4	4
k_0	5	10	15	20	5	10	15	20	5	10	15	20
g	2	2	2	2	2	2	2	2	2	2	2	2
min	24	49	74	99	19	39	59	79	14	29	44	79
max	25	50	75	100	20	40	60	80	15	30	45	80

Table 4: Numerical results, changing k_0

The concrete counterexamples produced by the Uppaal model checker can easily be transformed into parametric counterexamples: we just replace the concrete values of the timing constants by parameters and collect the constraints on these parameters that are imposed by the guards and invariants that occur in the

N	2			3			4		
C	6	6	6	6	6	6	6	6	6
n	4	4	4	4	4	4	4	4	4
k_0	10	10	10	10	10	10	10	10	10
g	2	3	4	2	3	4	2	3	4
min	49	24	16	39	19	13	29	14	9
max	50	25	17	40	20	14	30	15	10

Table 5: Numerical results, changing g

counterexample execution. Inspection of the counterexamples for the WSN protocol, which one can rerun step by step in the simulator, reveals that there are essentially three different scenarios that may lead to a state in which the network is not synchronized. In order to describe these scenarios parametrically at an abstract level, we need a bit of notation. We say that $s \in \{0, \ldots, \mathsf{C}-1\}$ is a *transmitting* slot, notation $\mathsf{TX}(s)$, if there is some node i that is transmitting in s, that is,

$$\mathsf{TX}(s) \quad \Leftrightarrow \quad (\exists i \in \mathsf{Nodes})(\mathsf{tsn}[i] = s).$$

We let $\mathsf{PREV}(s)$ denote the nearest transmitting slot that precedes s (cyclically). Formally, function $\mathsf{PREV} : \{0, \ldots, \mathsf{C}-1\} \rightarrow \{0, \ldots, \mathsf{C}-1\}$ is defined by

$$\mathsf{PREV}((s+1)\%\mathsf{C}) \;=\; \begin{cases} s & \text{if } \mathsf{TX}(s) \\ \mathsf{PREV}(s) & \text{otherwise} \end{cases} \tag{4}$$

We write $\mathsf{D}(s)$ to denote the number of slots visited when going from $\mathsf{PREV}(s)$ to s, that is, $\mathsf{D}(s) = (s - \mathsf{PREV}(s))\%C$. We define $\mathsf{M} = \max_s \mathsf{D}(s)$ to be the maximal distance between transmitting slots. As we will see, M plays a key role in defining correctness.

3.1. Scenario 1: Fast Sender - Slow Receiver

In the first error scenario, a sending node is proceeding maximally fast whereas a receiving node runs maximally slow. The sender starts with the transmission of a message while the receiver is still in an earlier slot. The scenario is illustrated in Figure 8. It starts when the fast and the slow node receive a synchronization message. Immediately following receipt of this message (at the same point in time), the hardware clock of fast node ticks and the synchronizer resets this clock to $\mathsf{g}+1$. Now, in the worst case, it may take $\mathsf{M} \cdot \mathsf{k}_0 - 1$ ticks before the fast node is in its TX slot with its hardware clock equal to g. Since the hardware clock of the fast node ticks maximally fast, the length of the corresponding time interval is $(\mathsf{M} \cdot \mathsf{k}_0 - 1) \cdot \mathsf{min}$. The slow node will reach the TX slot of the fast node after $\mathsf{M} \cdot \mathsf{k}_0 - \mathsf{g}$ ticks. With a clock that ticks maximally slow, this may take $(\mathsf{M} \cdot \mathsf{k}_0 - \mathsf{g}) \cdot \mathsf{max}$ time. If $(\mathsf{M} \cdot \mathsf{k}_0 - \mathsf{g}) \cdot \mathsf{max}$ is greater than or equal to $(\mathsf{M} \cdot \mathsf{k}_0 - 1) \cdot \mathsf{min}$ then we may end up in a state where the network is no longer synchronized since the fast

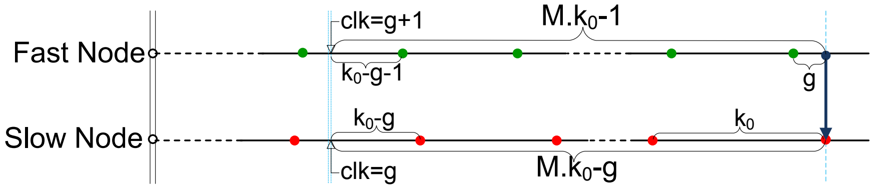

Figure 8: Scenario 1: Fast Sender - Slow Receiver

node is sending before the slow node has moved to the same slot. Hence, in order to exclude this scenario, we must have:

$$(\mathsf{M} \cdot \mathsf{k_0} - \mathsf{g}) \cdot \mathsf{max} \quad < \quad (\mathsf{M} \cdot \mathsf{k_0} - 1) \cdot \mathsf{min} \tag{5}$$

This constraint is consistent with the results in Table 2. Consider, for instance the first column. According to Uppaal the protocol is correct if $\mathsf{N} = 2$, $\mathsf{C} = 6$, $\mathsf{n} = 4$, $\mathsf{k_0} = 10$, $\mathsf{g} = 2$, $\mathsf{min} = 49$ and $\mathsf{max} = 50$. Since we assume that the two nodes are sending in the first two slots of a frame, it is easy to see that $\mathsf{M} = 5$. Now we can verify that

$$(5 \cdot 10 - 2) \cdot 50 \;=\; 48 \cdot 50 \;=\; 2400 \;<\; 2401 \;=\; 49 \cdot 49 \;=\; (5 \cdot 10 - 1) \cdot 49$$

However, if we increase the clock drift slightly by setting min and max to 48 and 49, respectively, then the protocol fails according to Uppaal. And indeed

$$(5 \cdot 10 - 2) \cdot 49 \;=\; 48 \cdot 49 \;=\; 2352 \;=\; 49 \cdot 48 \;=\; (5 \cdot 10 - 1) \cdot 48$$

Instead of the lower bound min and the upper bound max on the time between clock ticks, we sometimes find it convenient to consider the ratio

$$\rho = \frac{\mathsf{min}}{\mathsf{max}}$$

Since $0 < \mathsf{min} \leq \mathsf{max}$, it follows that ρ is contained in the interval $(0, 1]$. The following elementary lemma turns out to be quite useful.

Lemma 1. *Constraint (5) is equivalent to* $\mathsf{g} > (1 - \rho) \cdot \mathsf{M} \cdot \mathsf{k_0} + \rho$.

This implies that the worst case scenario occurs when the distance between TX slots is maximal: if the constraint holds for M it also holds when we replace M by a smaller value.

Example 1 (The Chess implementation). Constraint (5) allows us to infer a lower bound on the guard time g. In the current implementation of the protocol by Chess [3], a quartz crystal oscillator is used with a clock drift rate θ of at most 20 ppm. This means that

$$\rho \;=\; \frac{1 - \theta}{1 + \theta} \;=\; \frac{1 - 20 \cdot 10^{-6}}{1 + 20 \cdot 10^{-6}} \approx 0,99996$$

In the Chess implementation, one time frame lasts for about 1 second. It consists of $\mathsf{C} = 1129$ slots and each slot consists of $\mathsf{k_0} = 29$ clock ticks. The number of active slots is small ($\mathsf{n} = 10$). A typical value for M is $\mathsf{C} - \mathsf{n} = 1119$. Hence

$$\mathsf{g} > (1 - \rho) \cdot \mathsf{M} \cdot \mathsf{k_0} + \rho \approx 0,00004 \cdot 1119 \cdot 29 + 0,99996 \;=\; 2.298$$

Thus, according to our theoretical model, a value of $\mathsf{g} = 3$ should suffice. Chess actually uses a guard time of 9. Of course one should realize here that our model is overly simplified and, for instance, does not take into account (uncertainty in) message delays and partial connectivity. We will see that these restrictions greatly influence the minimal guard time.

3.2. Scenario 2: Fast Receiver - Slow Sender - before transmission

In our second error scenario, a receiving node runs maximally fast whereas a sending node proceeds maximally slow. The receiving node already leaves the slot in which it should receive a message from the sender before the sender has even started transmission. This scenario is illustrated in Figure 9. Again, the scenario starts when the fast and the slow node receive a synchronization message. But now the node that has to send the next message runs maximally slow. It sends this message after $\mathsf{M} \cdot \mathsf{k_0}$ ticks have occurred, which takes $\mathsf{M} \cdot \mathsf{k_0} \cdot \mathsf{max}$ time. Meanwhile, the fast node has made maximal progress: immediately after receipt of the first synchronization message (at the same point in time), the hardware clock of the fast node

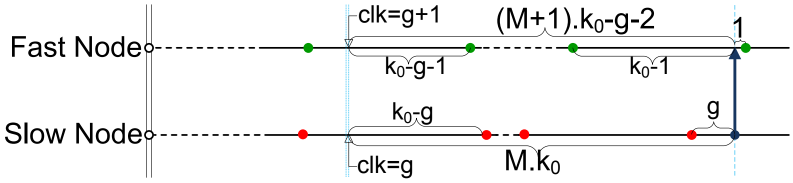

Figure 9: Scenario 2: Fast Receiver - Slow Sender - before transmission

ticks and the synchronizer resets this clock to $\mathsf{g}+1$. Already after $(\mathsf{k_0}-\mathsf{g}-1)\cdot\mathsf{min}$ time the node proceeds to the next slot. Another $(\mathsf{M}\cdot\mathsf{k_0}-1)\cdot\mathsf{min}$ time units later the fast node sets its clock to $\mathsf{k_0}-1$ and is about to leave the slot in which the slow node will send a message. If the slow node starts transmission after this point it is too late: after the next clock tick the fast node will increment its slot counter and the network is no longer synchronized. In order to exclude the second scenario, the following constraint must hold:

$$\mathsf{M}\cdot\mathsf{k_0}\cdot\mathsf{max} \quad < \quad ((\mathsf{M}+1)\cdot\mathsf{k_0}-\mathsf{g}-2)\cdot\mathsf{min} \tag{6}$$

Also this constraint can be rewritten:

Lemma 2. *Constraint (6) is equivalent to* $\mathsf{g} < (1-\frac{1}{\rho})\cdot\mathsf{M}\cdot\mathsf{k_0}+\mathsf{k_0}-2$*.*

Thus constraint (6) imposes an upper bound on guard time g. Since in practice one will always try to minimize the guard time in order to save energy, this constraint is only of theoretical interest. If we fill in the values of Example 1, we obtain $\mathsf{g} < 25.8$, which is close to the slot length $\mathsf{k_0} = 29$.

3.3. Scenario 3: Fast Receiver - Slow Sender - during transmission

Our third scenario involves a fast receiver and a slow sender. The receiver moves to a new slot while the sender is still transmitting a message. Figure 10 illustrates the scenario. As in the previous scenarios, the hardware clock of the fast node is set to $\mathsf{g}+1$ immediately after receipt of the synchronization message.

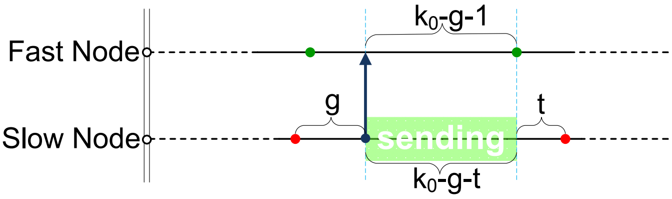

Figure 10: Scenario 3:Fast Receiver- Slow Sender - during transmission

To exclude this scenario, the following condition should be satisfied:

$$(\mathsf{k_0}-\mathsf{g}-\mathsf{t})\cdot\mathsf{max} \quad < \quad (\mathsf{k_0}-\mathsf{g}-1)\cdot\mathsf{min} \tag{7}$$

Essentially, constraint (7) provides a lower bound on t: to rule out the scenario in Figure 10, the sender should wait long enough before proceeding to the next slot.

Lemma 3. *Constraint (7) is equivalent to* $\mathsf{t} > (1-\rho)(\mathsf{k_0}-\mathsf{g})+\rho$*.*

If we fill in the values of Example 1 with g set to 3, we obtain $\mathsf{t} > 1.001$. Hence a value of $\mathsf{t} = 2$ should suffice. Hence, for the simple case of a static network with full connectivity and no uncertainty in message delays, we only need to reserve 5 clock cycles for guard and tail time together. In Section 6, we will see that for different network topologies indeed much larger values are required.

4. Proving Sufficiency of the Constraints

In this section, we outline our proof that the three constrains derived in Section 3 are sufficient to ensure synchronization in networks with full connectivity. We first present the key invariants used in the proof and then discuss the formalization of the full proof using Isabelle/HOL.

4.1. Invariants

We start our proof by stating some elementary invariants.

Lemma 4. *For any network with full connectivity the following invariant assertions hold, for all reachable states and for all $i \in \mathsf{Nodes}$:*

$$
\begin{array}{rcll}
 & & 0 \le \mathsf{x}_i \le \mathsf{max} & (8)\\
 & & 0 \le \mathsf{clk}[i] < \mathsf{k_0} & (9)\\
 & & 0 \le \mathsf{csn}[i] < \mathsf{C} & (10)\\
\mathsf{GO\_SEND}_i & \Rightarrow & \mathsf{x}_i = 0 & (11)\\
\mathsf{GO\_SEND}_i & \Rightarrow & \mathsf{csn}[i] = \mathsf{tsn}[i] & (12)\\
\mathsf{GO\_SEND}_i & \Rightarrow & \mathsf{clk}[i] \in \{\mathsf{g}, \mathsf{g}+1\} & (13)\\
\mathsf{SENDING}_i & \Rightarrow & \mathsf{csn}[i] = \mathsf{tsn}]i] & (14)\\
\mathsf{SENDING}_i & \Rightarrow & \mathsf{g} \le \mathsf{clk}[i] < \mathsf{k_0} - \mathsf{t} & (15)
\end{array}
$$

Invariants (8)-(10) assert that the state variables indeed take values in their intended domains: clock variables stay within the (real-valued) range $[0, \mathsf{max}]$, hardware clocks stay within the integer range $[0, \mathsf{k_0})$, and current slot numbers stay within the integer range $[0, \mathsf{C})$. Invariants (11)-(15) directly follow from the definitions of the individual automata in the network. For invariant (13), observe that since the tick?-transition from WAIT to GO_SEND may synchronize with the tick?-transition from S1 to S0, the value of $\mathsf{clk}[i]$ in $\mathsf{GO\_SEND}_i$ is potentially $\mathsf{g}+1$.

In order to be able to state more interesting invariants, we introduce two auxiliary global history (or ghost) variables. Clock y records the time that has elapsed since the last synchronization message (or the beginning of the protocol). Variable last records the last slot in which a synchronization message has been sent (initially $\mathsf{last} = -1$). Figure 11 shows the version of the $\mathbf{WSN}(i)$ automaton obtained after adding these variables. The only change is that upon occurrence of a synchronization $\mathsf{start\_message}[i]!$ clock y is reset to 0 and variable last is reset to $\mathsf{csn}[i]$. We first state a few basic invariants which restrict the values of the new variables.

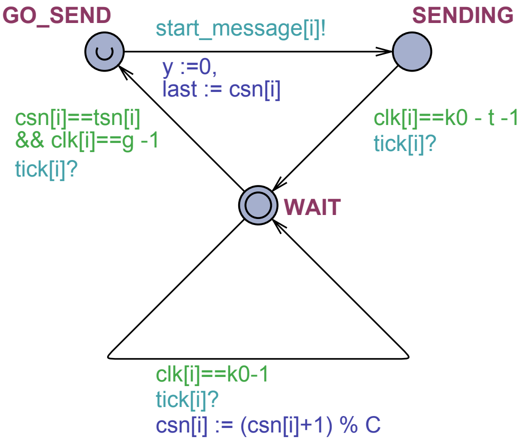

Figure 11: $\mathbf{WSN}(i)$ with history variables

Lemma 5. *For any network with full connectivity the following invariant assertions hold, for all reachable states and for all $i \in \mathsf{Nodes}$:*

$$\begin{aligned} & 0 \leq \mathsf{y} && (16)\\ & -1 \leq \mathsf{last} < \mathsf{C} && (17)\\ \mathsf{S1}_i \quad \Rightarrow \quad & \mathsf{y} \leq \mathsf{x}_i && (18)\\ \mathsf{last} = -1 \quad \Rightarrow \quad & \mathsf{S0}_i && (19) \end{aligned}$$

Invariant (16) says that y is always nonnegative, and invariant (17) says that last takes values in the integer domain $[-1, \mathsf{C}-1)$. If the system is in $\mathsf{S1}_i$ then a synchronization occurred after the last clock tick (invariant (18)), and if the system is in $\mathsf{S0}_i$ then no synchronization occurred yet (invariant (19)).

The key idea behind our correctness proof is that, given the local state of some node i and the value of last, we can compute the number $c(i)$ of ticks of i's hardware clock that has occurred since the last synchronization. Since we know the minimal and maximal clock speeds, we can then derive an interval that contains the value of y, the amount of real-time that has elapsed since the last synchronization. Next, given the value of y, we can compute an interval that contains the value of $c(j)$, for arbitrary node j. Once we know the value of $c(j)$, this gives us some information about the local state of node j. Through these correspondences, we are able to infer that if node i is sending the slot number of i and j must be equal.

Formally, for $i \in \mathsf{Nodes}$, the state function $c(i)$ is defined by

$$\begin{aligned} c(i) \quad = \quad & \textbf{if } \mathsf{last} = -1 \textbf{ then } \mathsf{clk}[i] \textbf{ else}\\ & \qquad \textbf{if } \mathsf{S1}_i \textbf{ then } 0 \textbf{ else}\\ & \qquad\qquad ((\mathsf{csn}[i] - \mathsf{last})\%\mathsf{C}) \cdot \mathsf{k_0} + \mathsf{clk}[i] - \mathsf{g}\\ & \qquad \textbf{fi}\\ & \textbf{fi} \end{aligned}$$

If there has been no synchronization yet ($\mathsf{last} = -1$) then $c(i)$ is just equal to the hardware clock $\mathsf{clk}[i]$. If the synchronizer is in location $\mathsf{S1}_i$, then we know that there has been no tick since the last synchronization, so $c(i)$ is set to 0. Otherwise, $c(i)$ is $\mathsf{k_0}$ times the number of slots since the last synchronization, incremented by the number of ticks in the current slot, minus g to take into account that the hardware clock has been reset to $\mathsf{g}+1$ after the last synchronization.

We can now state the main invariant result from this section.

Theorem 6. *Assume constraints (5), (6) and (7),* $3 \leq \mathsf{N}$ *and assume that some node transmits in the initial slot.*[1] *Then for any network with full connectivity the following invariant assertions hold, for all reachable*

[1] The last two assumptions have been made for convenience. We claim that they are not needed, even though they are used in our proof.

states and for all $i, j \in \mathsf{Nodes}$*:*

$$
\begin{aligned}
& & & \mathsf{y} \le c(i) \cdot \mathsf{max} + \mathsf{x}_i && (20)\\
c(i) > 0 & & \Rightarrow\; & \mathsf{y} \ge (c(i) - 1) \cdot \mathsf{min} + \mathsf{x}_i && (21)\\
\mathsf{csn}[i] = \mathsf{tsn}[i] \wedge (\mathsf{clk}[i] < \mathsf{g} \vee \mathsf{GO\_SEND}_i) & & \Rightarrow\; & \mathsf{last} \ne \mathsf{csn}[i] && (22)\\
\mathsf{csn}[i] = \mathsf{tsn}[i] \wedge \mathsf{clk}[i] = \mathsf{g} & & \Rightarrow\; & (\mathsf{GO\_SEND}_i \vee \mathsf{SENDING}_i) && (23)\\
\mathsf{csn}[i] = \mathsf{tsn}[i] \wedge \mathsf{clk}[i] > \mathsf{g} & & \Rightarrow\; & \mathsf{last} = \mathsf{csn}[i] && (24)\\
\mathsf{SENDING}_i & & \Rightarrow\; & \mathsf{csn}[i] = \mathsf{csn}[j] = \mathsf{last} && (25)\\
\mathsf{GO\_SEND}_i & & \Rightarrow\; & \mathsf{csn}[i] = \mathsf{csn}[j] \wedge \mathsf{clk}[i] = \mathsf{g} && (26)\\
\mathsf{last} \ne -1 \wedge \mathsf{last} \ne \mathsf{PREV}(\mathsf{csn}[i]) & & \Rightarrow\; & (\mathsf{TX}(\mathsf{csn}[i]) \wedge \mathsf{last} = \mathsf{csn}[i]) && (27)\\
\mathsf{TX}(\mathsf{csn}[i]) \wedge \mathsf{clk}[i] = \mathsf{k_0} - 1 & & \Rightarrow\; & \mathsf{last} = \mathsf{csn}[i] && (28)\\
\mathsf{S1}_i & & \Rightarrow\; & \mathsf{clk}[i] < \mathsf{k_0} - 1 \wedge \mathsf{last} = \mathsf{csn}[i] && (29)\\
& & & c(i) \ge 0 && (30)\\
\mathsf{last} = -1 & & \Rightarrow\; & \mathsf{csn}[i] = 0 && (31)
\end{aligned}
$$

Proof. By induction, using the invariants from Lemma's 4 and 5.

For a manual proof see `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/HSV09/`.

Invariants (20) and (21) are the key invariants that relate the values of $c(i)$ and y. Invariant (25) implies that the network is synchronized. This is the main correctness property we are interested in. All the other invariants in Theorem 6 are auxiliary assertions, needed to make the invariant inductive.

4.2. On the formal proof

The manual proof of the invariants from the previous subsection has been fully checked using the proof assistant Isabelle [19]. Below we make some general remarks about the formalization and discuss some of the subtleties we encountered. Our main motivation for discussing some of the proof detailts is that this sheds light on the type of reasoning that will be necessary in order to completely automate the verification.

The length of the Isabelle/HOL proof is about 5300 lines, whereas the manual proof is around 1000 lines. Formal proofs are usually longer than their manual counterpart. Wiedijk [20, 21] proposes the *De Bruijn factor* as a way to quantify this difference. This factor basically compares the size of two proof files, compressed using the Unix utility *gzip*. Wiedijk [20] observes that the average De Bruijn factor is about 4. In our case, we obtain 4.58. This is a bit larger than usual, since our formal proof includes the definition of the Uppaal model and its semantics, which are not included in the manual proof. We also need to define all the invariants (about 500 lines). In the manual proof, the 12 basic invariants defined in Lemmas 4 and 5 are all disposed of by the word "trivial". The formal proof is indeed straightforward but still occupies about 440 lines.

Key aspects of the Isabelle formalization are (1) an alternative definition of function PREV and a proof of lemmas showing particular properties of it, and (2) a formalization of the claim that there are at least three transmitting slots per frame. Common to these two issues is the introduction of the largest slot number in which a message is transmitted. This is the maximum of function tsn and is obtained for node i_{max}. The properties we need are basic facts like $\mathsf{PREV}(s)$ cannot be s or that in the idle period of a frame $\mathsf{PREV}(s)$ equals the transmitting slot of i_{max}, i.e., $\mathsf{tsn}[i_{\mathsf{max}}]$. Altogether, the definition of PREV, the introduction of i_{max}, the formal proof that there are at least three transmitting slots, and the proof of basic properties about these notions occupy about 600 lines.

In the remainder of this section, we first formally introduce i_{max}. Then, we rephrase the definition of function PREV and prove a sequel of properties of that function. After that, we formalize the claim that there are at least three transmitting slots. Finally, we illustrate the formal proof by two simple but representative examples.

4.2.1. Definition of i_{max} *and* PREV

As shown in Figure 1, a frame is composed of an active period and an idle period. In the active period, there are slots where a node is transmitting and the other nodes are listening, and also slots where no node is sending and all nodes are listening. Consequently, there is a last slot in which a message is emitted. Let i_{max} be the node that is transmitting in this slot. This transmitting node maximizes function tsn:

$$\mathsf{TX}_{\mathsf{max}}(i_{\mathsf{max}}) \equiv \mathsf{TX}(\mathsf{tsn}[i_{\mathsf{max}}]) \wedge \forall i \neq i_{\mathsf{max}}.\mathsf{tsn}[i_{\mathsf{max}}] > \mathsf{tsn}[i] \tag{32}$$

The formal definition of function PREV in Isabelle slightly differs from Equation 4. The combination of modulo and the incrementation in the argument does not translate to Isabelle, where functions must be total and proved to terminate. We basically remove the modulo and considers unbounded frames. We still have the assumption that function tsn returns a natural number strictly less than n. The first basic invariants then prove that parameters take values in their intended domain. Function PREV is the recursive function below:

Definition 2.

$$\begin{array}{lcl} \mathsf{PREV}(0) & = & \mathsf{tsn}[i_{\mathsf{max}}] \\ \mathsf{PREV}(s+1) & = & \textbf{if } \mathsf{TX}(s) \textbf{ then } s \textbf{ else } \mathsf{PREV}(s) \end{array}$$

4.2.2. Properties of PREV

In the formal proof, we need a sequel of properties showing the structure of a frame. The next lemma asserts that function PREV is constant during the idle period, that is, if slot s is transmitting and all slots from s to y are not transmitting, then $\mathsf{PREV}(y)$ is slot s.

Lemma 7. $(\mathsf{TX}(s) \wedge y > s \geq 0 \wedge (\forall z.s < z < y \Rightarrow \neg\mathsf{TX}(z))) \Rightarrow \mathsf{PREV}(y) = s$

Proof. By induction on s.

From this above lemma it directly follows that after the last transmitting slot, function PREV equals this slot:

Lemma 8. $\forall y > \mathsf{tsn}[i_{\mathsf{max}}].\mathsf{PREV}(y) = \mathsf{tsn}[i_{\mathsf{max}}]$

Proof. By definition i_{max} is such that there is no transmitting slot after it. We use this fact to instantiate Lemma 7 above.

We prove that the previous slot of slot s is strictly less than s. Because of the cyclic nature of a frame, this is only true if $s > 0$.

Lemma 9. $s > 0 \Longrightarrow \mathsf{PREV}(s) < s$

Proof. By induction on s.

Another useful lemma asserts that the "PREV" of a transmitting slot cannot be $\mathsf{tsn}[i_{\mathsf{max}}]$.

Lemma 10. $(s > 0 \wedge \mathsf{TX}(s)) \Longrightarrow \mathsf{PREV}(s) < \mathsf{tsn}[i_{\mathsf{max}}]$

Proof. From $\mathsf{TX}(s)$ we obtain j such that $\mathsf{tsn}[j] = s$. By definition of i_{max} we have $\mathsf{tsn}[j] \leq \mathsf{tsn}[i_{\mathsf{max}}]$. Using Lemma 9, we obtain $\mathsf{PREV}(s) < s$. Hence $\mathsf{PREV}(s) < \mathsf{tsn}[i_{\mathsf{max}}]$.

4.2.3. At least three sending nodes

In the informal case study description [3], it is assumed that for each node there is a transmission slot. Translated to the setting of our model, this means that tsn is a total function from nodes to slots. Interestingly, the Isabelle formalization revealed that the assumption that tsn is total is never used in the proof.[2] The only assumption that we make is that there are at least three sending nodes.

In our formalization, we introduce a predicate $\mathsf{TX}_n(i)$ which states that for node i there exists a slot s that equals the transmitting slot of node i, that is, node i is a transmitting node. Predicate $\mathsf{TX}_n(i)$ complements predicate $\mathsf{TX}(s)$ defined earlier. Predicate $\mathsf{TX}_n(i)$ is defined as follows:

Definition 3. $\mathsf{TX}_n(i) = \exists s.\mathsf{tsn}[i] = s$

The assumption that there are at least three transmitting slots is formalized by assuming that predicate TX_n holds for nodes 0 to 2.

$$\forall i \leq 2.\mathsf{TX}_n(i) \tag{33}$$

We derive two important facts: $\mathsf{tsn}[i_{\mathsf{max}}]$ is at least 2, and between slot number 0 and slot number $n-1$ there is at least one transmitting slot.

Lemma 11. $\mathsf{tsn}[i_{\mathsf{max}}] \geq 2 \wedge \exists s.0 < s < n-1 \wedge \mathsf{TX}(s)$

Proof. The first part is trivial. Function tsn assigns different slots to different nodes. Together with Equation 33 we can derive three distinct nodes – say i, j, k – with distinct transmitting slots ($\mathsf{tsn}[i]$, $\mathsf{tsn}[j]$, $\mathsf{tsn}[k]$). We do a case analysis on their different possible orderings (e.g., $\mathsf{tsn}[i] < \mathsf{tsn}[j] < \mathsf{tsn}[k]$). By definition a slot number is not greater than $n-1$ and positive. Consequently, the "tsn" in the middle of the ordering is strictly positive and strictly less than $n-1$. This shows the second term of our conclusion.

A consequence of Lemma 11 is that function PREV is at least one for all slots not smaller than $n-1$.

Lemma 12. $s \geq n-1 \Longrightarrow \mathsf{PREV}(s) \geq 1$

Proof. We consider two cases. If $s > n-1$, then $s > \mathsf{tsn}[i_{\mathsf{max}}]$. Moreover there is no transmitting slot between s and $n-1$. So, from Lemma 7 we obtain $\mathsf{PREV}(s) > \mathsf{tsn}[i_{\mathsf{max}}] > 1$. If $s = n-1$, then we know from Lemma 11 that there is at least one transmitting node between slot 0 and $n-1$ and $\mathsf{PREV}(s)$ is then at least equal to this slot.

4.2.4. Proof samples

In the remainder of this section, we present two examples that show some of the subtleties in the proof. These example illustrates why some of the lemmas introduced earlier are needed, e.g., Lemma 10 and Lemma 11.

Example 2. The situation of our first proof sample is pictured in Figure 12. This situation appeared in the proof of Invariant 21 and 23 of Theorem 6. It involves nodes k and an arbitrary different node j. Node k is sending in its current slot number, i.e. we have $\mathsf{csn}[k] = \mathsf{tsn}[k]$ and $\mathsf{TX}(\mathsf{csn}[k])$. The last transmitting slot (depicted in the gray slot) is the previous transmitting slot of both nodes j and k.

$$\mathsf{PREV}(\mathsf{csn}[k]) = \mathsf{PREV}(\mathsf{csn}[j]) = \mathsf{last} \tag{34}$$

The goal is to prove that these two nodes agree on the current slot number, i.e., that $\mathsf{csn}[k] = \mathsf{csn}[j]$.

The formal proof needs a case analysis on the relative positions of $\mathsf{csn}[j]$ and $\mathsf{csn}[k]$. Assume $\mathsf{csn}[j] < \mathsf{csn}[k]$. Node k is in a later slot. Because of the cyclic nature of frames, we must consider two cases: $\mathsf{csn}[j] = 0$ and $\mathsf{csn}[j] > 0$.

[2] This observation may have practical implications. Our results suggest that, at least in certain situations, if nodes have nothing to say they may in fact remain silent.

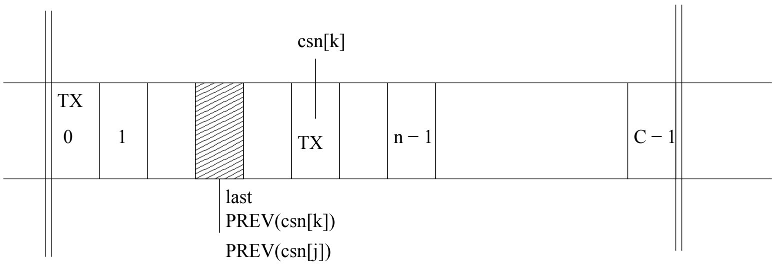

Figure 12: Proof example

If $\mathsf{csn}[j] = 0$, by definition of PREV we have $\mathsf{PREV}(\mathsf{csn}[j]) = \mathsf{tsn}[i_{\mathsf{max}}]$, and $\mathsf{PREV}(\mathsf{csn}[k]) = \mathsf{tsn}[i_{\mathsf{max}}]$ also. From Lemma 10 we have $\mathsf{PREV}(\mathsf{csn}[k]) < \mathsf{csn}[k]$. So, we have $\mathsf{tsn}[i_{\mathsf{max}}] < \mathsf{csn}[k] = \mathsf{tsn}[k]$. By definition of i_{max}, this is clearly impossible.

The case when $\mathsf{csn}[k] < \mathsf{csn}[j]$ is similar. We know that $\mathsf{TX}(\mathsf{csn}[k])$, so k is transmitting and has to be in the active region, i.e., $\mathsf{csn}[k] < n$. We do not have such information for $\mathsf{csn}[j]$ and need to consider extra cases: $\mathsf{csn}[j] < n$ and $\mathsf{csn}[j] \geq n$.

Example 3. We now illustrate the proof of Invariant 22 in Theorem 6. This invariant assumes that node i is in a transmitting slot. We have $\mathsf{csn}[i] = \mathsf{tsn}[i]$, hence $\mathsf{TX}(\mathsf{csn}[i])$. It also considers that node i is either in state WAIT with $\mathsf{clk}[i] < \mathsf{g}$) or in state $\mathsf{GO\_SEND}$. (By basic Invariant 15 node i cannot be in state $\mathsf{SENDING}$.) In brief, node i is about to send a message. The conclusion asserts that the last slot with a synchronization is not the current slot ($\mathsf{last} \neq \mathsf{csn}[i]$). Before the first synchronization, last is negative ($\mathsf{last} = -1$) and the conclusion holds as any csn is a nonnegative number (basic Invariant 10). Before a $\mathsf{start\_message}$ action, node i is in state $\mathsf{GO\_SEND}$ with its clock equal to g. Variable $\mathsf{clk}[i]$ is not modified by this action. Hence the invariant is trivially true in the target state because its premises are false. The case of a tick action is more complicated.

We only consider the end of the current slot ($\mathsf{csn}[i]$) The situation is as follows. Node i is in state WAIT with its clock counting the last tick of a slot ($\mathsf{clk}[i] = \mathsf{k_0} - 1$). After the tick action the clock is reset to 0, a new slot starts ($\mathsf{csn}'[i] = (\mathsf{csn}[i] + 1)\%C$), and other variables are left unchanged, in particular $\mathsf{last}' = \mathsf{last}$. The last transmitting slot is the previous transmitting slot of i ($\mathsf{last} = \mathsf{PREV}(\mathsf{csn}[i])$). We illustrate the case where $\mathsf{csn}[i] = 0$ or $\mathsf{csn}[i] = n - 1$.

The latter implies that $\mathsf{csn}'[i] = 0$. Our conclusion rewrites to $\mathsf{PREV}(n - 1) \neq 0$. This directly follows from the fact that PREV is at least one (Lemma 12).

If $\mathsf{csn}[i] = 0$, we have $\mathsf{csn}'[i] = 1$. Our conclusion rewrites to $\mathsf{PREV}(0) \neq 1$. By definition, $\mathsf{PREV}(0) = \mathsf{tsn}[i_{\mathsf{max}}]$ and our conclusion follows from the first term of Lemma 11 ($\mathsf{tsn}[i_{\mathsf{max}}] \geq 2$).

5. Uppaal Analysis Results: Networks with 4 Nodes

In the two previous sections, we studied the correctness of our clock synchronization protocol for networks with full connectivity. In practice, however, wireless sensor networks are rarely fully connected. A fully parametric analysis of the protocol for arbitrary network topologies will be quite involved. In order to illustrate the some of the complications, we wrote a small script to explore all possible network topologies with 4 nodes. As explained in Section 2, we can easily model arbitrary network topologies in Uppaal by appropriate instantiation of the $\mathsf{neighbor}$ function. For parameters $\mathsf{k_0} = 15$, $\mathsf{C} = 6$ and $\mathsf{n} = 4$, we used Uppaal to find out for each topology for which guard time and $\mathsf{min}/\max$ ratio the synchronization property was satisfied. This took 30 hours for the $4^6 = 4096$ possible topologies. In Table 6, we summarize the results of our model checking experiments for the connected networks of Figure 13 in which communication is symmetric. As in Section 3, min and max in this table are the smallest consecutive natural numbers for which the model with the values assigned to C, n, $\mathsf{k_0}$ and g is synchronized. As expected, topology number 6, requires the highest guard time of all networks with 4 nodes. We observe that the more connected the network is, the lower the guard time can be.

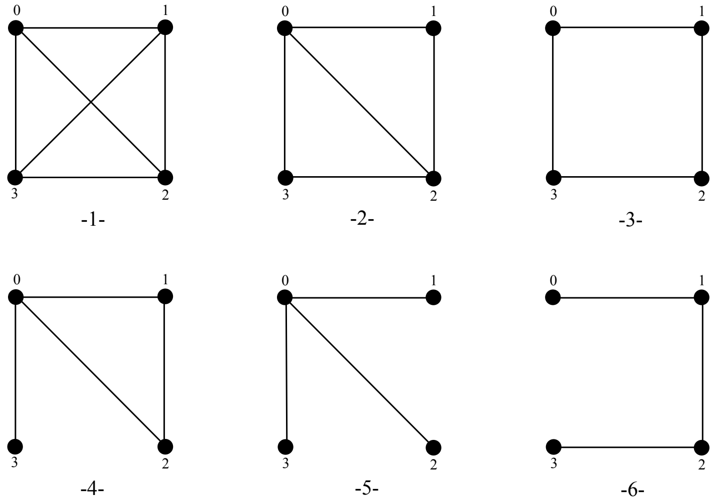

Figure 13: Connected networks with 4 nodes

ID	Diameter	Time Slot Numbers	Guard Time	min/max Ratio
1	1	any	2	44/45
2	2	{0,1,2,3}	3	58/59
		{0,2,1,3}	3	73/74
3	2	{0,1,2,3}	3	58/59
		{0,2,1,3}	3	73/74
4	2	{0,1,2,3}	3	58/59
		{0,1,3,2}	3	43/44
		{0,3,1,2}	3	73/74
5	2	any	3	88/89
6	3	{0,1,2,3}	4	88/89
		{0,3,1,2}	3	43/44

Table 6: Analysis results for the networks of Figure 13

When communication is not symmetric, that is, it may occur that some node A receives messages from a node B but not vice versa, the clock synchronization behavior becomes highly unpredicatable and depends on time slot numbers assigned to each node. Table 7 summarizes our analysis results for the networks depicted in Figure 14. Surprisingly, network 1 allows for smaller guard times than network 3, even though it has fewer links.

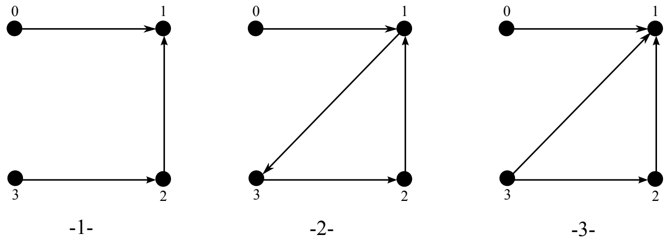

Figure 14: Some directed network topologies with 4 nodes

Time Slot Numbers	ID	Guard Time	min/max Ratio
{0,1,2,3}	1	2	74/75
	2	4	83/84
	3	3	38/39
{1,0,2,3}	1	2	45/46
	2	4	87/88
	3	3	43/44
{2,1,3,0}	1	2	30/31
	2	3	87/88
	3	3	73/74
{1,2,0,3}	1	2	74/75
	2	not synchronized at all	
	3	2	74/75

Table 7: Analysis results for the networks of Figure 14

6. Uppaal Analysis Results: Line Topologies

Since our experiments indicate that, among the symmetric topologies, line topologies have the worst clock synchronization behavior, we report on the model checking using UPPAAL of some further instances of the protocol that involve line topologies, that is, connected networks in which each node is connected to exactly two other nodes, except for two nodes that only have a single neighbor.

We defined a 3-node network with line topology in UPPAAL and checked the behavior of the system for different variable valuations. It turns out that, unlike the fully connected network with three nodes (see Table 2), the network will not always remain synchronized for $\mathsf{g} = 2$, even when the clocks are perfect. Table 8 lists some of our verification results. On the left we give the results for a line network of size 3 and on the right those for a clique network of size 3. If we compare these results then we see that, in order to keep the network synchronized, the hardware clocks in a line topology must be more accurate than the hardware clocks in a fully connected network (i.e., the min/max ratio must be closer to 1 if we want the network to be synchronized). Intuitively, the reason is that in a line topology the frequency of synchronization for each node is less than that in a fully connected network. In order to maintain synchronization, a line topology requires more accurate hardware clocks and a larger guard time. We claim that, for a fixed value of the guard time, the network may become unsynchronized if we keep increasing the number of nodes. In fact, we claim that for a line topology of size N, the guard time g should be at least N.

C	6	8	10	12
n	4	4	4	4
$\mathsf{k_0}$	10	10	10	10
g	3	3	3	3
min	58	78	98	118
max	59	79	99	119

C	6	8	10	12
n	4	4	4	4
$\mathsf{k_0}$	10	10	10	10
g	3	3	3	3
min	19	29	39	49
max	20	30	40	50

Table 8: Results for line network of size 3 (left) and for clique network of size 3 (right)

Model checking of synchronization for line topology entails exploring a state space that grows exponentially with the number of nodes. In order to reduce the state space, we considered only networks with perfect clocks. However, even with perfect clocks, UPPAAL can only handle networks with at most 8 nodes. Table 9 shows the resource usage of UPPAAL required for model checking of networks with line topologies. We used a Sun Fire X4440 machine with 4 Opteron 8356 2.3 Ghz quad-core processors and 128 Gb DDR2-667 memory. One processor on this machine needs about half an hour to establish that a line network with 8 nodes is synchronized if the guard time is 8.

Nodes	g	Collision	Time	Memory
2	1	yes	0.008 s	240852 KB
	2	no	0.039 s	240852 KB
3	2	yes	0.160 s	240852 KB
	3	no	0.200 s	240852 KB
4	3	yes	1.007 s	240852 KB
	4	no	1.012 s	240852 KB
5	4	yes	2.570 s	240852 KB
	5	no	2.587 s	240852 KB
6	5	yes	17.000 s	240852 KB
	6	no	18.006 s	240852 KB
7	6	yes	163.154 s	326892 KB
	7	no	173.922 s	336672 KB
8	7	yes	1624.481 s	2328572 KB
	8	no	1681.874 s	2451884 KB

Table 9: CPU time and memory usage of Uppaal for line networks of different sizes

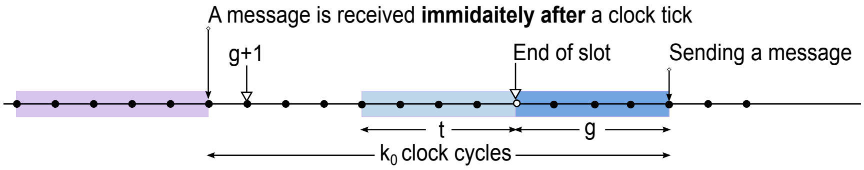

Figure 15: Maximum distance between two consecutive clock synchronization events

The reason why we run into state space explosions even in a setting with perfect clocks, is that race conditions are possible involving arrival of messages and ticking of hardware clocks. As a result even a network with perfect clocks will not necessarily remain synchronized for any parameter valuation. Figures 15 and 16 illustrate how race conditions may affect the time interval between two synchronization events in our model. (For simplicity, we assume $\mathtt{g} = \mathtt{t}$.) We consider the case where a node is the receiver in one slot and the sender in the next slot. We know that the sender sends a message when the value of its clock equals $\mathtt{g}$, and that the receiver resets its clock counter to $\mathtt{g}+1$ at the first clock tick after receiving the message. Figure 15 shows that a synchronization signal is received immediately *after* a clock tick at the receiver. In this scenario, the receiver waits a full clock cycle before resetting its clock counter to $\mathtt{g}+1$. Figure 16 illustrates a different scenario in which a synchronization signal is received immediately *before* the

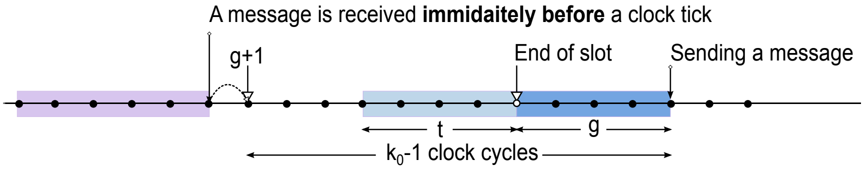

Figure 16: Minimum distance between two consecutive clock synchronization events

receiver clock ticks and the receiver immediately resets its clock counter to $\mathsf{g}+1$. We see that the length of the time interval between two synchronization events in the first scenario is one clock cycle longer than that in the second scenario.

We will now show that in a line network of size N and with guard time $\mathsf{g}=\mathsf{N}-1$, there is a reachable state in which the network is no longer synchronized. Thus synchronization of line networks can only be ensured if $\mathsf{g} \geq \mathsf{N}$. In our examples, $\mathsf{tsn}[i] = i\%3$, that is, the transmission slot number of node i equals i modulo 3. Note that, for line topologies, this allocation of transmission slot numbers satisfies the conditions (2) and (3) defined at the end of Section 2. Figures 17, 18 and 19 illustrate three abstract error scenarios, extracted from concrete counterexamples produced by UPPAAL, resulting in a loss of synchronization. Figure 17 applies to the case in which N modulo 3 equals 0, Figure 18 to the case in which N modulo 3 equals 1, and Figure 19 to the case in which N modulo 3 equals 2. We explain the example of Figure 17 in detail. The other two scenarios are similar.

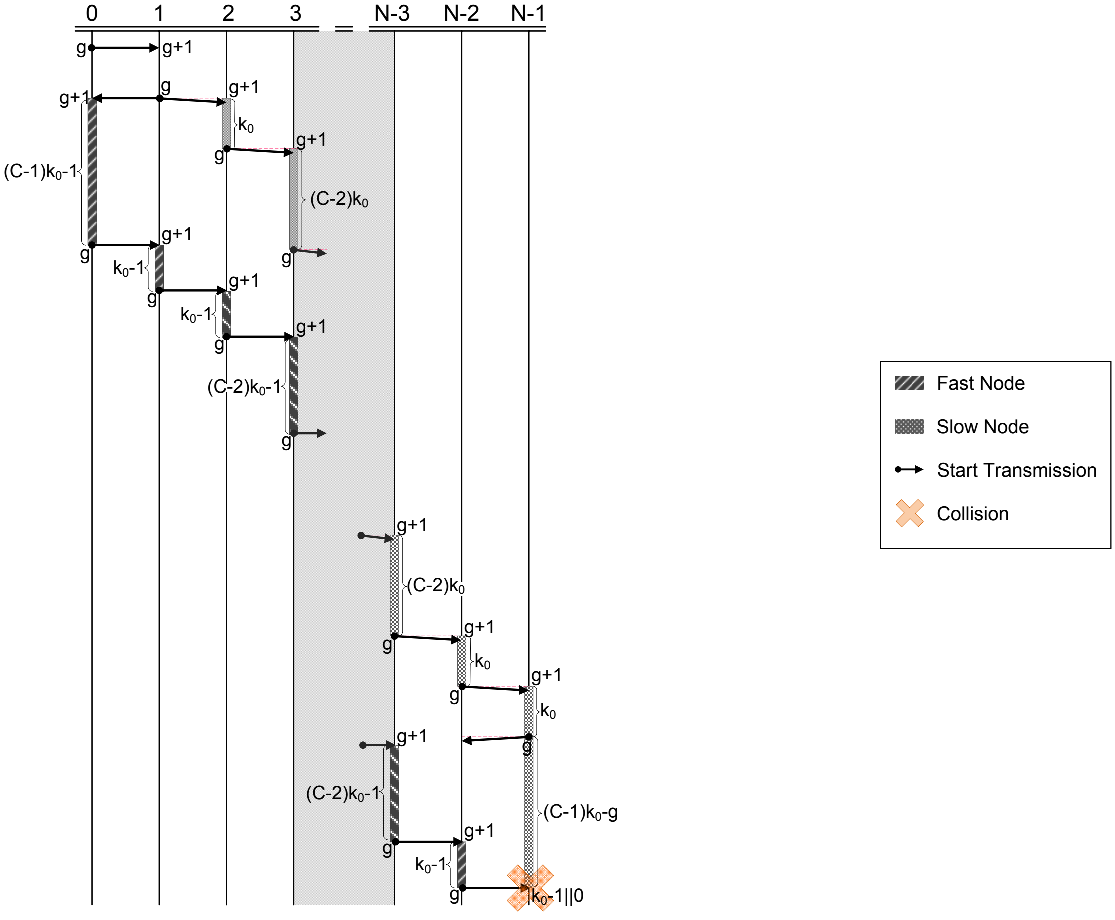

Figure 17: Error scenario for line topologies when $\mathsf{N} \mod 3 = 0$

The scenario of Figure 17 consists of two "staircases". One "fast" staircase has steps with minimum time between synchronizations (using the mechanism of Figure 16), where a synchronization signal is received immediately before the receiver clock ticks and the receiver resets its clock counter to $\mathsf{g}+1$ immediately, while the other "slow" staircase has steps with the maximum time between synchronizations (using the mechanism of Figure 15), where a synchronization signal is received immediately after the receiver clock ticks, and it takes an additional clock tick before resetting the clock is reset to $\mathsf{g}+1$. Both staircases start from the same point, viz. when node number 1, the second node in the line, sends messages to its neighboring nodes 0 and 2. After $\mathsf{N}-1$ steps the two staircases join again when node $\mathsf{N}-2$ tries to communicate with node $\mathsf{N}-1$. At that point, node $\mathsf{N}-2$ has gone through g time units since its previous synchronization and is about to send a message to node $\mathsf{N}-1$. However, node $\mathsf{N}-1$ is about to make a clock tick and enter its new time slot, which is convenient for receiving the message from its neighbor. Synchronization is lost when node $\mathsf{N}-2$ starts sending before node $\mathsf{N}-1$ ticks.

This proof, in sum, shows that for each network of line topology in which guard time is less than N clock cycles, based on the given error scenarios, the network may fail to keep synchronization. Accordingly, to guarantee synchronization in a network of line topology with N nodes, guard time should be equal to or grater than N.

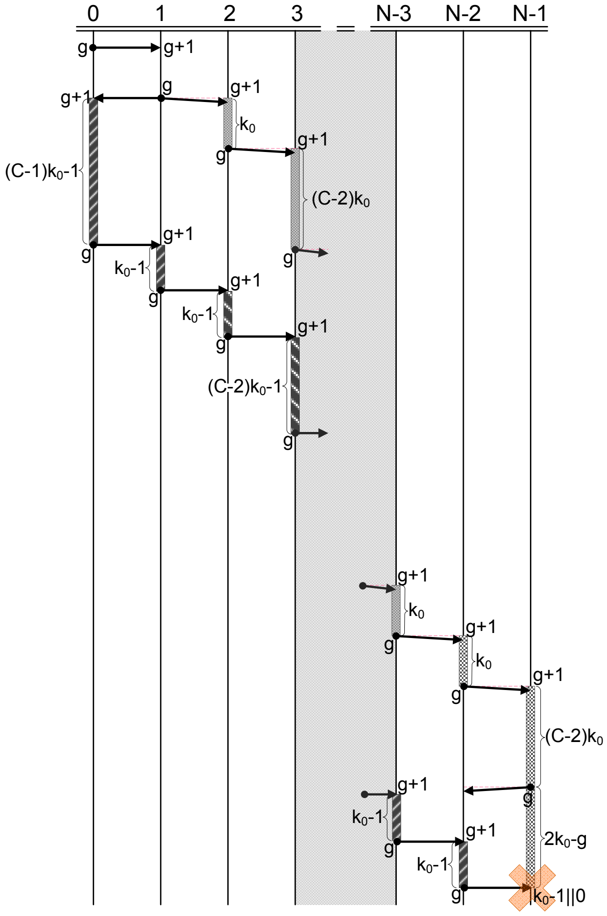

Figure 18: Error scenario when $\mathsf{N} \mod 3 = 1$

7. Conclusions and Related Work

Wireless sensor networks constitute a potentially very important but also extremely challenging application area for formal methods. As we have seen in this paper, even the analysis of a basic clock synchronization algorithm for an industrial WSN platform turns out to be quite difficult. Formal analysis of the gossip layer is a largely unexplored research field [6].

Using timed automata model checking, we discovered some interesting error scenarios for line topologies: for any instantiation of the parameters, the protocol will eventually fail if the network grows. We also succeeded in presenting a parametric verification for the very restrictive case of cliques (network with full

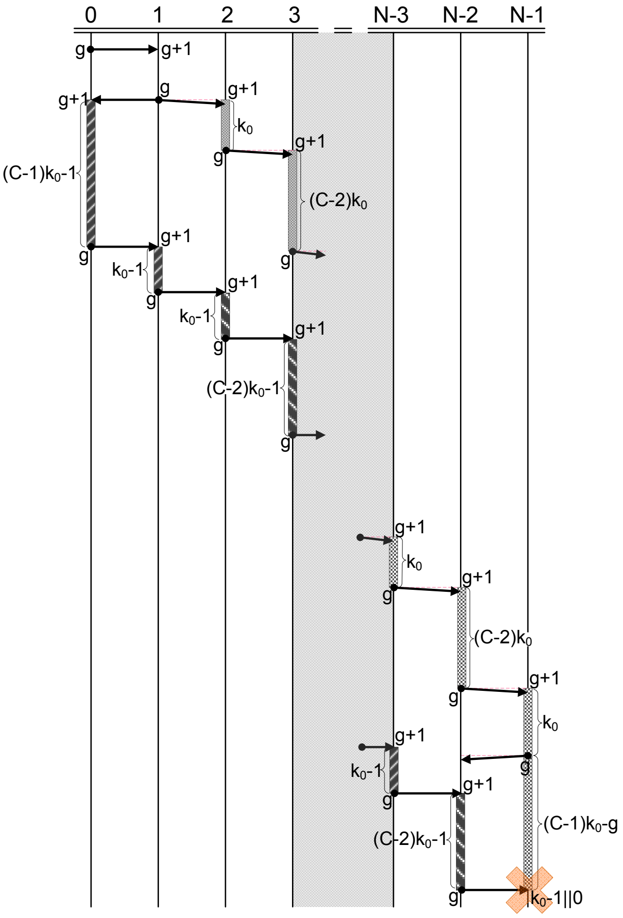

Figure 19: Error scenario when $\mathsf{N} \mod 3 = 2$

connectivity). We used model checking to find the key error scenarios that underly the parameter constraints for correctness, and theorem proving to check the correctness of our manual invariant proof. Despite its limitations, Uppaal proved to be indispensable for the formal analysis of the Chess protocol. Modelling the protocol in terms of timed automata is natural, the graphical user interface helped to visualize the models, the simulator was of great help during the initial validation of the model, and the ability of Uppaal to generate counterexamples and to replay them in the simulator helped us to find the parameter constraints that are needed for correctness. Since Uppaal does not support parametric verification, we proved the sufficiency of the constraints manually. But also here Uppaal was helpful: by checking the validity of various invariants for instances of the model with Uppaal, we obtained confidence in their correctness before embarking on the (long and tedious) invariant proofs.

Using state-of-the-art model checking technology, we have only been able to analyze models of some really small networks. In order to carry out our analysis we had to make some drastic simplifying assumptions. Clearly, there is a lot of room for improving timed automata model checking technology, enabling the analysis of larger and dynamic network topologies. Nevertheless, we conclude that the ability of model checkers to find worst-case error scenarios appears to be quite useful in this application domain. In particular, we succeeded to reproduce error scenarios — found using Uppaal by exploring simple models of small networks — in real implementations of larger networks [22].

In practical applications of WSNs, cliques rarely occur and therefore our verification results should primarily be seen as a first step towards a correctness proof for arbitrary and dynamically changing network topologies. Nevertheless, these results give us an upper bound on allowable clock drift of a generic WSN. The use of simulations will be essential for providing additional insight into the robustness and usefulness of our algorithm, also because occasional flaws of the MAC layer protocol may be resolved by the redundancy of the gossip layer. However, we believe it is unlikely that simulation techniques will be able to produce worst case counterexamples, such as the example of Figure 17 that was produced by the model checker Uppaal. Work of [23] also shows that one has to be extremely careful in using the results of MANET simulators.

Methodologically, the approach of this paper is similar to our study of the Biphase Mark Protocol [24], which also uses Uppaal to analyze instances of the protocol and a theorem prover for the full parametric analysis. Theorem provers have been frequently and successfully applied for the analysis of clock synchronization protocols, see for instance [25, 26]. An interesting research challenge is to synthesize (or prove the correctness of) the parameter constraints for the Chess protocol fully automatically. Recently, some approaches have been presented by which, for instance, the (parametric) Biphase Mark Protocol can be verified fully automatically [27, 28]. Very interesting also is the recent work of [29, 30] on parameterized verification (using the SMT based tool MCMT) of networks with an arbitrary number of identical timed automata. However, we think that these approaches are not powerful enough (yet) to handle our WSN protocol in which the number N of sensor nodes is not fixed, and the parameter constraints and the length of the corresponding counterexamples depend on N. Moreover, in the case of our WSN algorithm the parameter constraints involve a product of three parameters, whereas the mentioned techniques can only handle linear constraints on the parameters. Finally, these new tools still lack the graphical user interface and expressive input language of Uppaal, which are key features that enable the application of formal methods in practice.

Wireless sensor networks algorithms pose many challenges for probabilistic model checkers and specification tools such as PRISM [31] and CaVi [32]. A first challenge is to make a more detailled, probabilistic model of radio communication that involves the possibility of message loss. Another challenge is to consider dynamic slot allocation. In this paper, we assumed a fixed slot allocation. However, in the actual implementation of Chess, a sophisticated probabilistic algorithm is used for dynamic slot allocation. Formal analysis of this algorithm would be very interesting. Another simplifying assumption made in this paper is that the network topology is fixed. A probabilistic model in which nodes may join or leave will be more realistic. Finally, the gossiping algorithms used by the Chess network are intrinsically probabilistic in nature. Practical obstacles for the application of probabilistic model checkers to the Chess case study are the limited expressivity of the input language of existing tool, and the small network sizes that can be handled. An interesting alternative approach for some of the problems in this area is the use of mean-field analysis, as proposed by Bakhshi et al [33].

Several other recent papers report on the application of Uppaal for the analysis of protocols for wireless sensor networks, see e.g. [34, 32, 35, 1]. In [36], Uppaal is also used to automatically test the power consumption of wireless sensor networks. Our paper confirms the conclusions of [34, 35]: despite the small number of nodes that can be analyzed, model checking provides valuable insight in the behavior of protocols for wireless sensor networks, insight that is complementary to what can be learned through the application of simulation, testing or theorem proving.

A fundamental open question is to establish an impossibility result along the lines of Fan & Lynch [9] for the setting of the Chess MAC layer in which clocks can be set back. The algorithm that we have analyzed in this paper appears to perform well for networks with a small diameter, but the results of Section 6 show that performance may degrade if the diameter increases. The question to find an algorithm that (a) achieves (near) optimal synchronization quality for the Chess MAC layer, and (b) is sufficiently simple to be implemented on today's WSN networks, is still wide open.

References

[1] F. Heidarian, J. Schmaltz, F. Vaandrager, Analysis of a clock synchronization protocol for wireless sensor networks, in: A. Cavalcanti, D. Dams (Eds.), Proceedings 16th International Symposium of Formal Methods (FM2009), Eindhoven, the Netherlands, November 2-6, 2009, Vol. 5850 of Lecture Notes in Computer Science, Springer, 2009, pp. 516–531.
URL `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/HSV09/`

[2] QUASIMODO, Case studies: Models, deliverable 5.5 from the FP7 ICT STREP project 214755 (QUASIMODO) (Jan. 2009).
URL http://www.quasimodo.aau.dk/Final/Deliverable-5.5.pdf

[3] QUASIMODO, Preliminary description of case studies, deliverable 5.2 from the FP7 ICT STREP project 214755 (QUASIMODO) (Jan. 2009).
URL http://www.quasimodo.aau.dk/Final/Deliverable-5.2.pdf

[4] A.-M. Kermarrec, M. v. Steen, Gossiping in distributed systems, SIGOPS Oper. Syst. Rev. 41 (5) (2007) 2–7. doi:http://doi.acm.org/10.1145/1317379.1317381.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, D. Terry, Epidemic algorithms for replicated database maintenance, in: PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, ACM, New York, NY, USA, 1987, pp. 1–12. doi:http://doi.acm.org/10.1145/41840.41841.

[6] R. Bakhshi, F. Bonnet, W. Fokkink, B. Haverkort, Formal analysis techniques for gossiping protocols, SIGOPS Oper. Syst. Rev. 41 (5) (2007) 28–36. doi:http://doi.acm.org/10.1145/1317379.1317385.

[7] F. Assegei, Decentralized frame synchronization of a TDMA-based wireless sensor network, Master's thesis, Eindhoven University of Technology, Department of Electrical Engineering (2008).

[8] B. Sundararaman, U. Buy, A. D. Kshemkalyani, Clock synchronization for wireless sensor networks: a survey, Ad Hoc Networks 3 (3) (2005) 281 – 323. doi:DOI: 10.1016/j.adhoc.2005.01.002.
URL http://www.sciencedirect.com/science/article/B7576-4FDMVS4-1/2/63ed40b032c6bf3ad5fca7fdcbe9e35a

[9] R. Fan, N. Lynch, Gradient clock synchronization, Distributed Computing 18 (4) (2006) 255–266.

[10] L. Lamport, Time, clocks and the ordering of events in distributed systems, Commun. ACM 21 (7) (1978) 558–564.

[11] L. Meier, L. Thiele, Gradient clock synchronization in sensor networks, Tech. Rep. 219, Computer Engineering and Networks Laboratory, ETH Zurich (2005).

[12] R. Pussente, V. Barbosa, An algorithm for clock synchronization with the gradient property in sensor networks, Journal of Parallel and Distributed Computing 69 (3) (2009) 261 – 265. doi:DOI: 10.1016/j.jpdc.2008.11.001.
URL http://www.sciencedirect.com/science/article/B6WKJ-4TYJTY5-1/2/38716d7d9f37c51edb8ba05e508fa2ce

[13] R. Tjoa, K. Chee, P. Sivaprasad, S. Rao, J. Lim, Clock drift reduction for relative time slot tdma-based sensor networks, in: Proceedings of the 15^{th} IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC2004), 2004, pp. 1042–1047.

[14] M. Schuts, F. Zhu, F. Heidarian, F. Vaandrager, Modelling clock synchronization in the Chess gMAC WSN protocol, in: S. A. et.al (Ed.), Proceedings Workshop on Quantitative Formal Methods: Theory and Applications (QFM'09), Vol. 13 of Electronic Proceedings in Theoretical Computer Science, 2009, pp. 41–54.
URL http://www.mbsd.cs.ru.nl/publications/papers/fvaan/chess09/

[15] F. Heidarian, A comment on Assegei's use of Kalman filters for clock synchronization, iCIS, Radboud University Nijmegen. Available at http://www.mbsd.cs.ru.nl/publications/papers/fvaan/HSV09/ (October 2010).

[16] QUASIMODO, Final report: case studies and tool integration, deliverable 5.10 from the FP7 ICT STREP project 214755 (QUASIMODO) (Apr. 2011).
URL http://www.quasimodo.aau.dk/Final/Deliverable-5.10.pdf

[17] G. Behrmann, A. David, K. Larsen, A tutorial on Uppaal, in: M. Bernardo, F. Corradini (Eds.), Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures, Vol. 3185 of Lecture Notes in Computer Science, Springer, 2004, pp. 200–236.

[18] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, M. Hendriks, Uppaal 4.0, in: Third International Conference on the Quantitative Evaluation of SysTems (QEST 2006), 11-14 September 2006, Riverside, CA, USA, IEEE Computer Society, 2006, pp. 125–126.

[19] T. Nipkow, L. Paulson, M. Wenzel, Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Vol. 2283 of Lecture Notes in Computer Science, Springer, 2002.

[20] F. Wiedijk, The "de Bruijn factor", webpage accessed October 2010, http://www.cs.ru.nl/~freek/factor/ (2010).

[21] N. d. Bruijn, A survey of the project Automath, in: J. Hindley, J. Seldin (Eds.), To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisms, Academic Press, 1980.

[22] QUASIMODO, Dissemination and exploitation, deliverable 5.6 from the FP7 ICT STREP project 214755 (QUASIMODO) (Feb. 2010).
URL http://www.quasimodo.aau.dk/Final/Deliverable-5.6.pdf

[23] D. Cavin, Y. Sasson, A. Schiper, On the accuracy of manet simulators, in: Proceedings of the 2002 Workshop on Principles of Mobile Computing, POMC 2002, October 30-31, 2002, Toulouse, France, ACM, 2002, pp. 38–43.

[24] F. Vaandrager, A. d. Groot, Analysis of a biphase mark protocol with Uppaal and PVS, Formal Aspects of Computing Journal 18 (4) (2006) 433–458. doi:http://dx.doi.org/10.1007/s00165-006-0008-1.
URL http://www.ita.cs.ru.nl/publications/papers/fvaan/BMP.html

[25] J. Rushby, A formally verified algorithm for clock synchronization under a hybrid fault model, in: PODC '94: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, ACM, New York, NY, USA, 1994, pp. 304–313. doi:http://doi.acm.org/10.1145/197917.198115.

[26] J. Schmaltz, A formal model of clock domain crossing and automated verification of time-triggered hardware, in: J. Baumgartner, M. Sheeran (Eds.), Formal methods in computer aided design, IEEE Computer Society, 2007, pp. 223–230.
URL http://doi.ieeecomputersociety.org/10.1109/FMCAD.2007.2

[27] G. M. Brown, L. Pike, Easy parameterized verification of biphase mark and 8N1 protocols, in: H. Hermanns, J. Palsberg (Eds.), TACAS, Vol. 3920 of Lecture Notes in Computer Science, Springer, 2006, pp. 58–72.

[28] S. Umeno, Event order abstraction for parametric real-time system verification, in: L. de Alfaro, J. Palsberg (Eds.), EMSOFT, ACM, 2008, pp. 1–10.

[29] A. Carioni, S. Ghilardi, S. Ranise, MCMT in the land of parameterized timed automata, in: J. Giesl, R. Hähnle (Eds.), 6th International Verification Workshop (VERIFY 2010), associated with IJCAR, Edinburgh, UK, July 20-21, 2010. Proceedings, 2010, available at `http://homes.dsi.unimi.it/∼ghilardi/allegati/verify.pdf`.

[30] S. Ghilardi, S. Ranise, MCMT: A model checker modulo theories, in: J. Giesl, R. Hähnle (Eds.), Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings, Vol. 6173 of Lecture Notes in Computer Science, Springer, 2010, pp. 22–29.

[31] M. Kwiatkowska, G. Norman, D. Parker, PRISM 2.0: A tool for probabilistic model checking, in: Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST04), IEEE Computer Society, 2004, pp. 322–323.

[32] A. Fehnker, M. Fruth, A. McIver, Graphical modelling for simulation and formal analysis of wireless network protocols, in: M. Butler, C. Jones, A. Romanovsky, E. Troubitsyna (Eds.), Methods, Models and Tools for Fault Tolerance, Vol. 5454 of Lecture Notes in Computer Science, Springer, 2009, pp. 1–24.

[33] R. Bakhshi, L. Cloth, W. Fokkink, B. Haverkort, Mean-field analysis for the evaluation of gossip protocols, in: QEST 2009, Sixth International Conference on the Quantitative Evaluation of Systems, Budapest, Hungary, 13-16 September 2009, IEEE Computer Society, 2009, pp. 247–256.

[34] A. Fehnker, L. van Hoesel, A. Mader, Modelling and verification of the lmac protocol for wireless sensor networks, in: J. Davies, J. Gibbons (Eds.), Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings, Vol. 4591 of Lecture Notes in Computer Science, Springer, 2007, pp. 253–272.

[35] S. Tschirner, L. Xuedong, W. Yi, Model-based validation of qos properties of biomedical sensor networks, in: L. de Alfaro, J. Palsberg (Eds.), Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT 2008, Atlanta, GA, USA, October 19-24, 2008, ACM, 2008, pp. 69–78.

[36] M. Woehrle, K. Lampka, L. Thiele, Exploiting timed automata for conformance testing of power measurements, in: J. Ouaknine, F. W. Vaandrager (Eds.), Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings, Vol. 5813 of Lecture Notes in Computer Science, Springer, 2009, pp. 275–290. doi:http://dx.doi.org/10.1007/978-3-642-04368-0.
URL `http://dx.doi.org/10.1007/978-3-642-04368-0`