Analysis of a Clock Synchronization Protocol for Wireless Sensor Networks*

Faranak Heidarian**, Julien Schmaltz, and Frits Vaandrager

Institute for Computing and Information Sciences
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
{F.Heidarian,J.Schmaltz,F.Vaandrager}@cs.ru.nl

Abstract. We study a clock synchronization protocol for the Chess WSN. 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. Finally, 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. This result suggests a variation of the fundamental result of Fan and Lynch on gradient clock synchronization, where the synchronization eventually fails as the network diameter grows, for a setting with logical clocks whose value may also decrease.

Key words: 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 exciting applications, ranging from monitoring of dikes to smart kindergartens, and from forest fire detection to monitoring of the Matterhorn. It is an active research area with numerous workshops and conferences arranged each year.

The Dutch company Chess is currently developing a WSN architecture using an epidemic (gossip) communication model [15]. Gossiping in distributed systems refers to the repeated probabilistic exchange of information between two

* Research supported by the European Community's Seventh Framework Programme under grant agreement no 214755 (QUASIMODO). A preliminary version of the model presented in this paper appeared in [14].

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

members [8, 6]. 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 [2]. 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 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

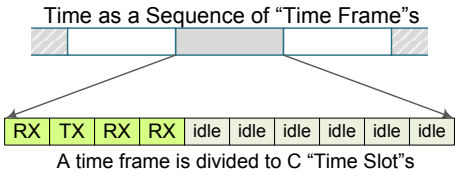

Fig. 1. The structure of a time frame

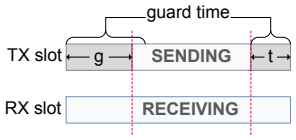

Fig. 2. TX and RX slots

either listening for incoming messages from neighboring nodes ("*RX*") or it is sending a message ("*TX*"). During idle slots a node is switched to energy saving mode. These 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, the number of active slots is typically much smaller than the total number of slots (less than 1% in the current implementation [15]). 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. The MAC protocol takes care that neighboring nodes have different TX slots.

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 clock synchronization in the Chess WSN. 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, a sender, ready with its transmission, waits a certain

amount of time for the receiver to be ready to receive messages, and it also waits for some time period at the end of its TX slot (see Figure 2). In the current

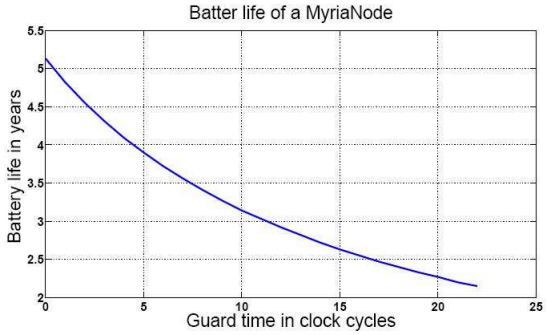

Fig. 3. battery life as a function of guard time

implementation, each slot consists of 29 clock cycles, out of which 18 cycles are used as guard time. Assegei [1] calculated how the battery life of a wireless sensor node is influenced by the guard time. Figure 3, taken from [1], summarizes these results. 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 Universal Coordinated Time (UTC). We refer to [18] 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 and Lynch [7] 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 [7], 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 [9] 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 and Lynch [7] assume that the rate of increase of each node's logical clock is at least $\frac{1}{2}$, at all times. Also Meier and Thiele [11], who adapt the work of Fan and Lynch to the setting of wireless sensor networks, make this assumption, but

then Pussente and Barbosa [13] assume this rate to be at least $\frac{1}{\mathsf{D}}$, 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. If we are willing to set back clocks now and then, we obtain even more efficient clock synchronization protocols.

The current implementation of the Chess WSN uses *Median*, an extension of an algorithm proposed by Tjoa et al [19]. 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 adjust their clock by the median of the phase error of their immediate neighbors. Assegei [1] points out that the performance of the Median algorithm decreases if the network becomes more dynamic, and proposes a variation of this algorithm that uses Kalman filters. 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 algorithm 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, robustness of our algorithm still needs to be explored further.

In Section 2, we model the algorithm using timed automata. Section 3 describes the use of the timed automata model checker Uppaal[4, 3] 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 assistent Isabelle [12]. Section 5 presents some result for the special case of line topologies: for any instantiation of the parameters, the protocol will eventually fail if the network grows. This result suggests a variation of the fundamental result of Fan and Lynch [7] on gradient clock synchronization for a setting with logical clocks whose value may also decrease. Section 6, finally, discusses related work and draws conclusions. Uppaal models and proofs for our paper are available at `http://www.ita.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.

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 [4, 3] and to `http://www.uppaal.com`.

We assume a finite, fixed set of wireless nodes $\mathsf{Nodes} = \{0, \ldots, \mathsf{N}-1\}$. The behavior of an individual node $i \in \mathsf{Nodes}$ is described by three timed automata **Clock**(i) (Section 2.1), **WSN**(i) (Section 2.2) and **Synchronizer**(i) (Section 2.3). Automaton **Clock**(i) models the hardware clock of node i, the **WSN**(i) automaton takes care of sending messages, and the **Synchronizer**(i) automaton resynchronizes the hardware clock of i upon receipt of a message. The complete protocol is modeled as a network that consists of timed automata **Clock**(i), **WSN**(i) and **Synchronizer**(i), for each $i \in \mathsf{Nodes}$.

Table 1 lists the parameters that are used in the model (constants in UPPAAL terminology), 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 $i \in \mathsf{Nodes}$	$0 \leq \mathsf{tsn}[i] < \mathsf{n}$
$\mathsf{k_0}$	number of clock ticks in a time slot	$0 < \mathsf{k_0}$
g	guard time	$0 < \mathsf{g}$
t	tail time	$0 < \mathsf{g},\ \mathsf{g} + \mathsf{t} + 2 \leq \mathsf{k_0}$
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) (Fig. 4) models 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 between clock ticks. Whenever x reaches the value min, the automaton enables an action $\mathsf{tick}[i]!$. Broadcast channel $\mathsf{tick}[i]$ is used to synchronize all activities within node i. The $\mathsf{tick}[i]!$ event 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 enable model checking, we reset the hardware clock after $\mathsf{k_0}$ ticks, that is, the clock takes integer values modulo $\mathsf{k_0}$ (we use UPPAAL's modulo operator %). This is not an essential modeling assumption and we can easily change this.

2.2 Wireless Sensor Node

The **WSN**(i) automaton, displayed in Figure 6, is the most important automaton in our model. It has three locations and four transitions. The automaton

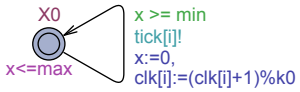

Fig. 4. Timed automaton **Clock**(i)

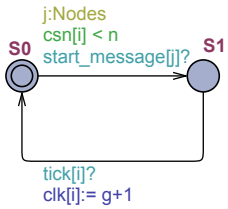

Fig. 5. Timed automaton **Synchronizer**(i)

uses an integer variable csn$[i]$, initially 0, to record its current slot number. The automaton stays in initial location WAIT until the current slot number of i equals the TX slot number of i (csn$[i]$ = 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, and then jumps back to location WAIT. At the end of each slot, i.e., when the ${\mathsf{k}_0}^{th}$ tick occurs, the automaton increments its current slot number (modulo C).

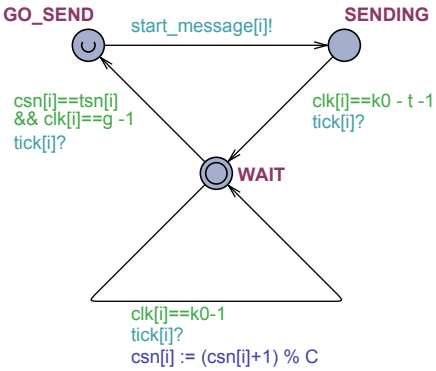

Fig. 6. Timed automaton **WSN**(i)

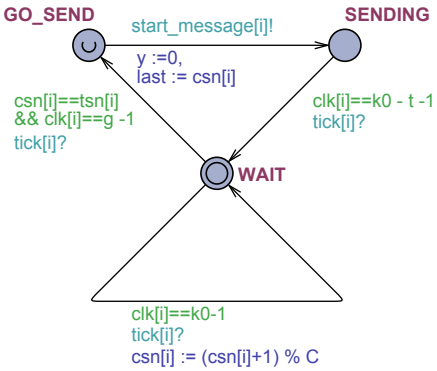

Fig. 7. WSN(i) with history variables

2.3 Synchronizer

The **Synchronizer**(i) automaton (Fig. 5) 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 some node j starts to transmit a new message, that is, until a

start_message[j]? event occurs. We use the UPPAAL select statement to nondeterministically select j. The automaton then moves to location S1, provided node i is active (csn[i] $<$ n). Remember that at the moment when the start_message[j]? event occurs, the hardware clock of node j, clk[j], has value g. Therefore, node i resets its own hardware clock clk[i] to g$+1$ upon occurrence of the first clock tick following the start_message[j]? event. The automaton then returns to its initial location S0.

Note that in our model there is no delay between sending and receipt of messages. Following [11], we assume delay uncertainties to be negligible, and we therefore eliminate the delays themselves from our analysis. When communication is infrequent, this is reasonable since the impact of clock drift dominates over the influence of delay uncertainties.

Automaton **Synchronizer**(i) (Fig. 4) has no constraint on the value of j, that is, we assume that node i can receive messages from all other nodes in the network. Hence the network has full connectivity. It is easy to generalize our model to a setting without full connectivity by adding a guard neighbor(i, j) to the transition from S0 to S1 that indicates that i is a direct neighbor of j.[1] For networks with full connectivity, we assume that all nodes have unique TX slot numbers:

$$(\forall i, j \in \mathsf{Nodes})(\mathsf{tsn}[i] = \mathsf{tsn}[j] \;\Rightarrow\; i = j).$$

For networks that are not fully connected, this assumption can be relaxed to the requirement that neighboring nodes have distinct TX slot numbers.

3 Uppaal Analysis Results

A wireless sensor network is called *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 assign different values to the parameters of the model and use UPPAAL to verify the property of Definition 1. Based on the outcomes (and in particular the counterexamples generated by UPPAAL) we try to derive general constraints. For networks with up to 4 nodes, UPPAAL is able to explore the state space within a few seconds.

[1] The neighbor(i, j) predicate does not have to be symmetric. In a wireless sensor network it may occur that i can receive messages from j, but not vice versa.

It turns out 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 at an abstract level, we need a bit of notation.

Let $s \in \{0, \ldots, \mathsf{C}-1\}$ be a slot. Then s 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 $\mathsf{PREV} : \{0, \ldots, \mathsf{C}-1\} \to \{0, \ldots, \mathsf{C}-1\}$ is defined by

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

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 can then start 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

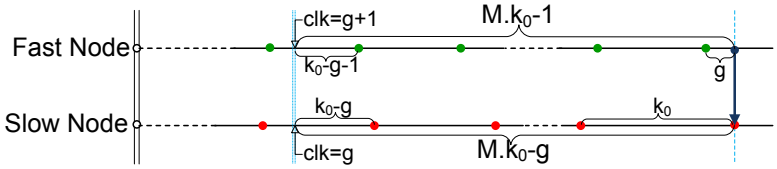

Fig. 8. Scenario 1: Fast Sender - Slow Receiver

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. To prevent the fast node from starting transmission before the slow node has moved to the same slot, we must have:

$$(\mathsf{M} \cdot \mathsf{k}_0 - \mathsf{g}) \cdot \mathsf{max} < (\mathsf{M} \cdot \mathsf{k}_0 - 1) \cdot \mathsf{min} \tag{2}$$

Rather than 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 (2) 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 (2) allows us to infer a lower bound on the guard time g. In the current implementation of the protocol by Chess [15], a quartz crystal oscillator is used with a clock drift rate θ of at most 20 ppm (parts per million). 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 the second 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. It 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 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

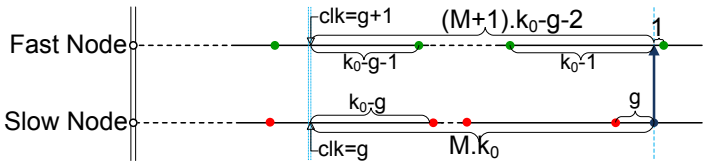

Fig. 9. Scenario 2: Fast Receiver - Slow Sender - before transmission

later the fast node sets its clock to $k_0 - 1$ 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:

$$M \cdot k_0 \cdot max < ((M + 1) \cdot k_0 - g - 2) \cdot min \tag{3}$$

Also this constraint can be rewritten:

Lemma 2. *Constraint (3) is equivalent to* $g < (1 - \frac{1}{\rho}) \cdot M \cdot k_0 + k_0 - 2$.

Thus constraint (3) 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 $g < 25.8$, which is close to the slot length $k_0 = 29$.

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

Our third scenario concerns 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 usual, the hardware clock of the fast node is set to $g + 1$ immediately after receipt of the synchronization message.

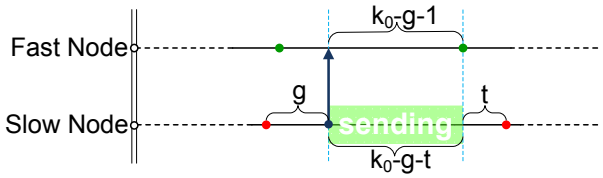

Fig. 10. Scenario 3:Fast Receiver- Slow Sender - during transmission

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

$$(k_0 - g - t) \cdot max < (k_0 - g - 1) \cdot min \tag{4}$$

Essentially, constraint (4) provides a lower bound on t: to rule out the scenario in Fig. 10, the sender should wait long enough before proceeding to the next slot.

Lemma 3. *Constraint (4) 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. 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 5, 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 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* $i \in \mathsf{Nodes}$*:*

$$0 \leq \mathsf{x}_i \leq \mathsf{max} \tag{5}$$

$$0 \leq \mathsf{clk}[i] < \mathsf{k_0} \tag{6}$$

$$0 \leq \mathsf{csn}[i] < \mathsf{C} \tag{7}$$

$$\mathsf{GO\_SEND}_i \Rightarrow \mathsf{x}_i = 0 \tag{8}$$

$$\mathsf{GO\_SEND}_i \Rightarrow \mathsf{csn}[i] = \mathsf{tsn}[i] \tag{9}$$

$$\mathsf{GO\_SEND}_i \Rightarrow \mathsf{clk}[i] \in \{\mathsf{g}, \mathsf{g}+1\} \tag{10}$$

$$\mathsf{SENDING}_i \Rightarrow \mathsf{csn}[i] = \mathsf{tsn}]i] \tag{11}$$

$$\mathsf{SENDING}_i \Rightarrow \mathsf{g} \leq \mathsf{clk}[i] < \mathsf{k_0} - \mathsf{t} \tag{12}$$

Invariants (5), (6) and (7) 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 (8)-(12) directly follow from the definitions of the automata in the network. For invariant (10), observe that since the $\mathsf{tick}?$-transition from WAIT to $\mathsf{GO\_SEND}$ may synchronize with the $\mathsf{tick}?$-transition from $\mathsf{S1}$ to $\mathsf{S0}$, the value of $\mathsf{clk}[i]$ in $\mathsf{GO\_SEND}_i$ is potentially $\mathsf{g}+1$.

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 7 shows the version of the $\mathbf{WSN}(i)$ automaton obtained after adding these variables. The only change is that upon occurrence of

a synchronization start_message$[i]$! clock y is reset to 0 and variable last is reset to csn$[i]$.

We first state a few basic invariants which restrict the values of the new variables.

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

$$0 \leq \mathsf{y} \tag{13}$$

$$-1 \leq \mathsf{last} < \mathsf{C} \tag{14}$$

$$\mathsf{S1}_i \Rightarrow \mathsf{y} \leq \mathsf{x}_i \tag{15}$$

$$\mathsf{last} = -1 \Rightarrow \mathsf{S0}_i \tag{16}$$

Invariant (13) says that y is always nonnegative and invariant (14) 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 (15)), and if the system is in $\mathsf{S0}_i$ then no synchronization occurred yet (invariant (16)).

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

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

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 1. *Assume constraints (2), (3) and (4) hold. Then for any network with full connectivity the following invariant assertions hold, for all reachable*

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

$$\mathsf{y} \leq c(i) \cdot \mathsf{max} + \mathsf{x}_i \tag{17}$$

$$c(i) > 0 \Rightarrow \mathsf{y} \geq (c(i) - 1) \cdot \mathsf{min} + \mathsf{x}_i \tag{18}$$

$$\mathsf{csn}[i] = \mathsf{tsn}[i] \wedge (\mathsf{clk}[i] < \mathsf{g} \vee \mathsf{GO\_SEND}_i) \Rightarrow \mathsf{last} \neq \mathsf{csn}[i] \tag{19}$$

$$\mathsf{csn}[i] = \mathsf{tsn}[i] \wedge \mathsf{clk}[i] = \mathsf{g} \Rightarrow (\mathsf{GO\_SEND}_i \vee \mathsf{SENDING}_i) \tag{20}$$

$$\mathsf{csn}[i] = \mathsf{tsn}[i] \wedge \mathsf{clk}[i] > \mathsf{g} \Rightarrow \mathsf{last} = \mathsf{csn}[i] \tag{21}$$

$$\mathsf{SENDING}_i \Rightarrow \mathsf{csn}[i] = \mathsf{csn}[j] = \mathsf{last} \tag{22}$$

$$\mathsf{GO\_SEND}_i \Rightarrow \mathsf{csn}[i] = \mathsf{csn}[j] \wedge \mathsf{clk}[i] = \mathsf{g} \tag{23}$$

$$\mathsf{last} \neq -1 \wedge \mathsf{last} \neq \mathsf{PREV}(\mathsf{csn}[i]) \Rightarrow (\mathsf{TX}(\mathsf{csn}[i]) \wedge \mathsf{last} = \mathsf{csn}[i]) \tag{24}$$

$$\mathsf{TX}(\mathsf{csn}[i]) \wedge \mathsf{clk}[i] = \mathsf{k}_0 - 1 \Rightarrow \mathsf{last} = \mathsf{csn}[i] \tag{25}$$

$$\mathsf{S1}_i \Rightarrow \mathsf{clk}[i] < \mathsf{k}_0 - 1 \wedge \mathsf{last} = \mathsf{csn}[i] \tag{26}$$

$$c(i) \geq 0 \tag{27}$$

$$\mathsf{last} = -1 \Rightarrow \mathsf{csn}[i] = 0 \tag{28}$$

Proof. By induction, using the auxiliary invariants from Lemma's 4 and 5. The manual proof is about 14 pages long.

Invariants (17) and (18) are the key invariants that relate the values of $c(i)$ and y. Invariant (22) implies that the network is synchronized. This is the key correctness property we are interested in. All the other invariants in Theorem 1 are auxiliary assertions, needed to make the invariant inductive.

5 Line Topologies

The line topology has the minimum connectivity. The number of clock synchronization events per time frame is the least possible value for all (connected) topologies. To maintain synchronization, we need more accurate hardware clocks and a larger guard time. We assert that for a fixed value of the guard time, the network fails to synchronize if one keeps increasing the number of nodes. We claim that for a line network of size N, guard time g should be at least N.

To reduce the state space of the UPPAAL analysis, we consider only networks with perfect clocks, in which clock drift is zero. In UPPAAL concurrent events are non-deterministically ordered. Depending on this choice, clock misalignment and loss of synchronization are possible.

Figure 11 shows a scenario extracted from a UPPAAL counter-example. This scenario shows that for a network of size N the guard time cannot be $\mathsf{N} - 1$.

The scenario consists of two "staircases". One "fast" staircase has stairs with the minimum width, where the sender transmits the synchronization signal immediately before the receiver experiences a tick event and the receiver resets its clock counter to $g + 1$ in no time as the transitions are urgent, while the other "slow" staircase has stairs with the maximum width, where the sender transmits the synchronization signal immediately after the receiver experienced

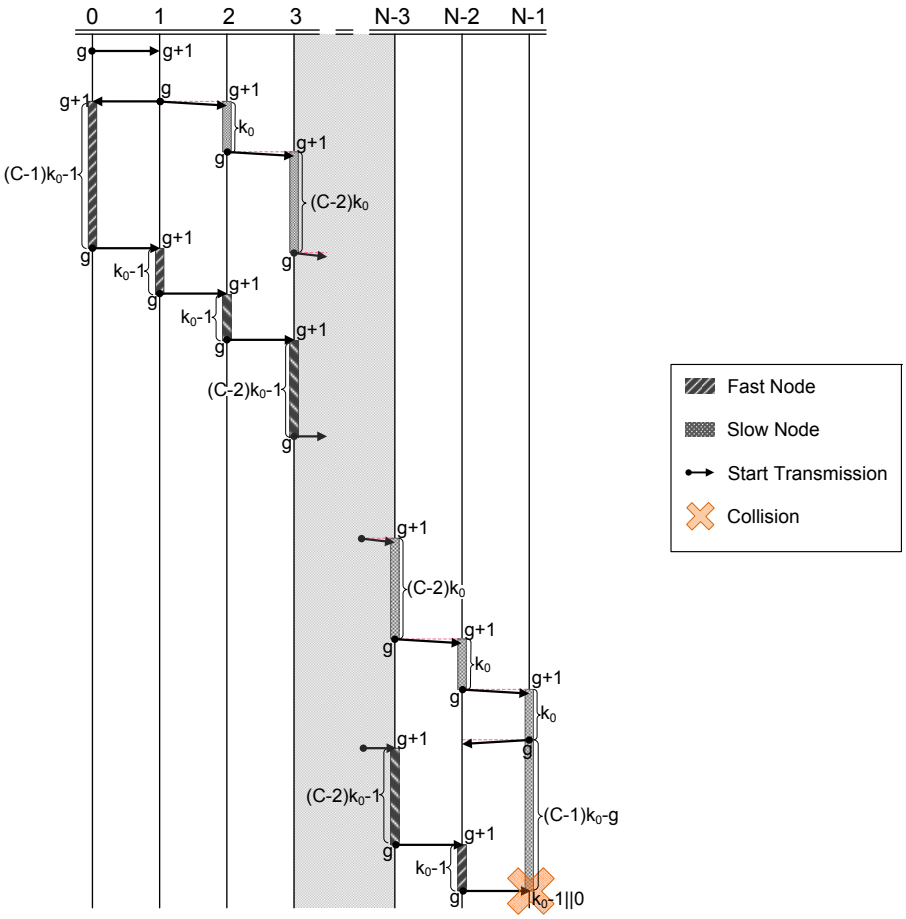

Fig. 11. An error scenario for line topologies

a tick event, so the receiver should wait a the duration of a tick before resetting its clock counter to $g + 1$. The staircases start from the same point, viz. when node number 1,the second node, tries to send messages to its neighbors, nodes 0 and 2. After $\mathsf{N} - 1$ steps, which takes a guard time period, 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 $\mathsf{N} - 1$. On the other hand, 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.

6 Conclusions and Related Work

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 believe that this error scenario is generic and may serve as a basis for a variation of the fundamental result of Fan and Lynch [7], reasserted by Locher and Wattenhofer [10], on gradient clock synchronization in a setting with logical clocks whose value may also decrease. We also succeeded in presenting a parametric verification for the very restrictive case of cliques (network with full 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. In practical applications of WSNs, cliques rarely occur and therefore our results should primarily be seen as a first step towards a correctness proof for arbitrary and dynamically changing network topologies. Nevertheless, these results could 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 simulation techniques will not be able to produce worst case counterexamples, such as the example of Figure 11 that was produced by the model checker Uppaal.

Methodologically, the approach of this paper is similar to our study of the Biphase Mark Protocol [21], 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 [16, 17]. 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 [5, 20]. However, we think these approaches are not powerful enough (yet) to handle the Chess protocol.

References

1. F.A. Assegei. Decentralized frame synchronization of a TDMA-based wireless sensor network. Master's thesis, Eindhoven University of Technology, Department of Electrical Engineering, 2008.
2. R. Bakhshi, F. Bonnet, W. Fokkink, and B. Haverkort. Formal analysis techniques for gossiping protocols. *SIGOPS Oper. Syst. Rev.*, 41(5):28–36, 2007.
3. G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Third International Conference on the Quantitative Evaluation of SysTems (QEST 2006),* 11-14 September 2006, Riverside, CA, USA, pages 125–126. IEEE Computer Society, 2006.
4. G. Behrmann, A. David, and K.G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems,*

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, volume 3185 of *LNCS*, pages 200–236. Springer, 2004.
5. G. M. Brown and L. Pike. Easy parameterized verification of biphase mark and 8n1 protocols. In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *LNCS*, pages 58–72. Springer, 2006.
6. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM.
7. R. Fan and N.A. Lynch. Gradient clock synchronization. *Distributed Computing*, 18(4):255–266, 2006.
8. A.-M. Kermarrec and M. van Steen. Gossiping in distributed systems. *SIGOPS Oper. Syst. Rev.*, 41(5):2–7, 2007.
9. L. Lamport. Time, clocks and the ordering of events in distributed systems. *Communications of the ACM*, 21(7):558–564, 1978.
10. T. Locher and R. Wattenhofer. Oblivious gradient clock synchronization. In *Distributed Computing*, volume 4167 of *LNCS*, pages 520–533. Springer, 2006.
11. L. Meier and L. Thiele. Gradient clock synchronization in sensor networks. Technical Report 219, Computer Engineering and Networks Lab., ETH Zurich, 2005.
12. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
13. R.M. Pussente and V.C. Barbosa. An algorithm for clock synchronization with the gradient property in sensor networks. *Parallel and Distributed Computing*, 69:261–265, 2009.
14. QUASIMODO. Case studies: Models, January 2009. Deliverable 5.5 from the FP7 ICT STREP project 214755 (QUASIMODO).
15. QUASIMODO. Preliminary description of case studies, January 2009. Deliverable 5.2 from the FP7 ICT STREP project 214755 (QUASIMODO).
16. J. Rushby. A formally verified algorithm for clock synchronization under a hybrid fault model. In *PODC '94: Thirteenth annual ACM symposium on Principles of distributed computing*, pages 304–313, New York, NY, USA, 1994. ACM.
17. J. Schmaltz. A formal model of clock domain crossing and automated verification of time-triggered hardware. In J. Baumgartner andM. Sheeran, editor, *Formal methods in computer aided design*, pages 223–230. IEEE Computer Society, 2007.
18. B. Sundararaman, U. Buy, and A.D. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281 – 323, 2005.
19. R. Tjoa, K.L. Chee, P.K. Sivaprasad, S.V. Rao, and J.G 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)*, pages 1042–1047, September 2004.
20. S. Umeno. Event order abstraction for parametric real-time system verification. In *EMSOFT*, pages 1–10. ACM, 2008.
21. F.W. Vaandrager and A.L. de Groot. Analysis of a biphase mark protocol with Uppaal and PVS. *Formal Asp. Comput.*, 18(4):433–458, December 2006.