Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 296 (2013) 261–277

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Modeling and Evaluation of Wireless Sensor Network Protocols by Stochastic Timed Automata

Fengling Zhang, Lei Bu, Linzhang Wang, Jianhua Zhao, Xin Chen, Tian Zhang, and Xuandong Li

State Key Laboratory for Novel Software Technology, Nanjing University, Jiangsu, P.R.China
Email: zfl@seg.nju.edu.cn, bulei@nju.edu.cn

Abstract

Wireless Sensor Networks (WSNs) are widely used in different kinds of environments. They may encounter lots of stochastic uncertainties and disturbances like message loss and node dynamics. Thus, it is critical to ensure the correctness of low level protocols in WSNs and evaluate their performance under different circumstances. In this paper, we propose a new method to analyze and evaluate WSN protocols based on stochastic timed automata and statistical model checking.
For modeling, the work flow of a WSN protocol can be modeled with classical timed automata. Then, to model the uncertainties such as message loss and node dynamics, which are common in realistic circumstances, the timed automata can be extended by stochastic transitions, resulting in the stochastic timed automata. For analysis, the correctness of the protocol can be answered by classical model checking on the timed automata, while the performance of the protocol under realistic environments can be evaluated by statistical model checking on the stochastic model. To illustrate the feasibility and scalability of the modeling and verification method presented in this paper, Timing-sync Protocol for Sensor Networks (TPSN) will be studied completely throughout the paper.

Keywords: Wireless Sensor Network Protocol, Modeling and Evaluation, Stochastic Timed Automata, Statistical Model Checking

1 Introduction

Nowadays, Wireless Sensor Networks (WSNs) have attracted world wide attention and have been used in military operations, medical care, environmental monitoring and protection etc [1]. All high level applications in WSNs are working based on their underlying protocols. To run these applications correctly and efficiently, the low level wireless sensor network protocols must be robust and reliable. In addition, devices in WSNs are usually difficult to change once deployed, so we must ensure that the protocols work well under their target environment in the design phase.

Currently, the approaches to examine the correctness of protocols mainly include simulation, testing and formal verification. Simulation and testing can be used to check large scale networks and discover errors, but they cannot guarantee to explore all the possible bugs in a system. As a result, researchers are trying to use formal modeling and verification to model the behaviors of a system and prove the correctness of the system under examination [2].

For modeling, as WSN protocols always have intensive connections with real time behaviors, timed automaton [5] is a natural modeling language for WSN systems. Studies like [16], [17], [18] have tried to use timed automata to model the work flow of the system. But classical timed automata can only model the behaviors of a system under ideal circumstance. As WSN systems can be deployed in any kinds of environments, they may encounter many kinds of uncertainties like message loss and node dynamics. Thus, it is important to introduce stochastic behaviors into timed automata to support the modeling of uncertainty behaviors to make the model more realistic.

For verification, model checking has been widely used to check the correctness of WSN protocols [15] [16] [19]. It can explore the full state space of the model for a protocol. Nevertheless, this technique is very expensive. It faces the notorious state-explosion problem and limits the scale of the networks that can be checked. As WSN systems always consist of dozens of nodes at least, classical model checking can not handle real-case WSN networks very well.

Fortunately, Statistical Model Checking (SMC) has recently been proposed as an alternative to avoid exhaustive exploration of the state-space of a model [3]. SMC is a simulation-based solution, which is less time and memory intensive than classical model checking [13]. The procedure of SMC is to generate enough sample execution paths for the system and then use the statistical hypothesis testing to decide whether the system satisfies the given property or not. SMC techniques can also be used to estimate the probability that a system satisfies a given property [3].

By combining these techniques, in this paper, we present a new method to model and evaluate WSN protocols by stochastic timed automata and statistical model checking. First of all, we propose to model WSN protocols with timed automata, to describe there work flow in ideal environment. For network dynamics, node failures and intermittent communication links which are common in WSNs [1, 6], we extend the timed automata with stochastic transitions. For example, in the extended stochastic timed automata, when a message is broadcasted, it has a probability to be lost instead of all the nodes receiving it successfully. Similarly, in real environment, a node may die and leave the network at any time, and it can resurrect and rejoin the network randomly as well.

For evaluation, we propose that the correctness of the design of a protocol can be verified by model checking on its ideal model. For example, whether there are deadlocks in the protocol, whether the protocol can achieve the function it claimed successfully and etc. Furthermore, by the technology of statistical model checking [8], we can check the performance of the protocol under different environments by adjusting the probability factors on the system, like what is the probability that the protocol can work correctly when the message loss probability is around 10%?

To illustrate the above method, we present a complete study of the modeling and evaluation of a well-known WSN protocol- Timing-sync Protocol for Sensor Networks (TPSN) [4], which is believed to have advantages of high precision and the suitability for multi-hop networks [4]. By using the modeling and verification method presented in this paper, we can find that, although the design of the protocol is correct in general, this protocol is extremely sensitive to environment and is not a suitable candidate when the system is supposed to work in harsh environments.

The paper is structured as follows. Section 2 gives the preliminary knowledge of this study, including the existing works in modeling and verification of WSN protocols, and a brief description to stochastic timed automata and their verification. In section 3, we present our method to model and evaluate a protocol with stochastic timed automata. The stochastic timed automata of a well-known WSN protocol and the respective evaluation are given in section 4. Section 5 describes the concluding remarks at last [1].

2 Background

2.1 Related Work

So far, there is no unified approach to model and verify WSN protocols. Related works in modeling and verification of WSN protocols are generally conducted in the following ways:

A well-known WSN protocol- Flooding Time Synchronization Protocol (FTSP) is widely studied by modeling and verification using timed automata in [14–16]. In [14], Kusy et al have considered complex environment in WSN systems, and proposed that a radio message would be dropped with a probability because of link failure. However, due to state space explosion, they only checked FTSP in network with 2 nodes without node and link failure. [15] has verified several important properties of FTSP and showed the network could converge to a single root node, and agree on a global time. But similar to [14], they did not consider link and node failure in their model. In [16], they modeled and verified networks consisting of 2-7 nodes. An error is described in a specific scenario where two nodes fail after the entire work is synchronized. Due to state space explosion, clock drift and link failure have not been introduced into their model.

In [17] and [18], Vaandrager et al also model a WSN protocol in the similar way. They verified the protocol with fully connected topology, and analyzed the counterexample given by the model checker. Nevertheless, their models does not incorporate all the features of the system neither, such as uncertain communication delays and unreliable radio communication.

In [19], a timed automaton of the Timing-sync Sensor Network Protocol (TPSN) is given. In this timed automaton, TPSN's work flow in ideal environment is described. Properties including whether all nodes can synchronize with the root node, and whether the clock drift between a node and the root are bounded are verified. They introduce integer clock to read and assign the value of local clocks. However,

[1] Part of the idea of this paper is published in our work-in-progress poster of abstract nature [22].

this operation increases the complexity of the model. The size of system they verified has only 3-5 nodes. Furthermore, they haven't considered message loss and node dynamics neither.

There are also methods to model protocols in other languages. Study [20] gives a method to check the performance of mobile WSNs by means of probabilistic model checking and PRISM [21]. They propose to model systems with the stochastic π-calculus and translate them into the PRISM language to check these properties. As most of the WSN protocols are time related, π-calculus cannot support this aspect very well. Furthermore, probabilistic model checking is very expensive which limits the size of the system that can be analyzed.

To conclude, we can see that, first, most of the modeling and evaluation works still treated WSN protocols as general protocols. Only a few considered to model the uncertainty behaviors, which are very common in WSN, into the system model. Second, most of the works only checked the correctness of the protocol, only a few of them considered the evaluation of the performance of the protocol under different profile or environment. Last but not least, as the mostly used analysis technique is model checking and/or probabilistic model checking which is very expensive, the size of the system that can be analyzed are limited. As the size of deployed WSN systems are always quite large, the scalability of current analysis technique can not handle real-case system very well.

2.2 Stochastic Modeling and Verification of Timed Automata

Timed automata [9] are widely used to model and analyze the behaviors of real time systems. A timed automaton is a finite state machine with a set of clocks to ensure adherence to strict timing constraints, such as execution times, response times and communication delays.

The simplest form of a constraint compares a clock value with a time constant [5]. Timed automata only allow Boolean combinations of simple constraints, i.e., for a set C of clock variables, the set $\Phi(C)$ of clock constraints δ is defined inductively by $\delta := x \leq n|n \leq x|\neg\delta|\delta_1 \wedge \delta_2$ where x is a clock in C and n is an integer constant. These clock variables are initiated with zero when the system is started, and then increase synchronously with the same rate. Clock variables can also be attached to locations as invariants. A location can be entered and stayed in only when all of its invariants are true.

Assume a finite alphabet Σ ranged over by a, b etc. standing for actions, a formal definition of timed automaton [9] Γ is a tuple $< L, l_0, E, I >$ where

- L is a finite set of locations (or nodes)
- $l_0 \in$ L is the initial location
- $E \in L \times \Phi(C) \times \Sigma \times 2^C \times L$ is the set of edges and
- $I : L \to \Phi(C)$ assigns invariants to locations

In a timed automaton, events are modeled as transitions. Clock constraints, i.e. guards on edges are used to restrict transitions of the automaton. A transition can only be taken when are values satisfy the guards labeled on the edge.

The theory of timed automata can be used to prove the correctness of real time systems [5]. Generally, two types of properties, liveness and safety, are concerned. As checking liveness is computationally expensive, the main effort of verifying a timed system focuses on checking the safety properties, which can be checked using reachability analysis by traversing the state-space of timed automata [9]. It is proved that the reachability verification of timed automata is decidable, still quite expensive though.

Timed automata can be extend to model stochastic transitions via weighted probabilistic branches, resulting in the stochastic timed automata. Recently, Statistical Model Checking (SMC) [12] is proposed to answer the numerical properties of stochastic systems. It simulates the system model repeatedly based on Monte Carlo simulation to generate enough sample execution paths, then relies on statistical algorithms, such as hypothesis testing [11], to get an estimate of the correctness of the entire design [8, 10].

Owing to the use of sample executions, the verification results can't be guaranteed to be correct all the time, but at least the likelihood of error can be bounded [8]. Let H_0 be the hypothesis that the property formula ϕ holds, and let H_1 be the alternative hypothesis. Two parameters α and β can be used to bound the probability of error, where α is the largest acceptable probability of H_1 given that H_0 holds, and the probability of accepting H_0 if H_1 holds should be no more than β [8]. Then, techniques like Wald's sequential probability ratio test [11] are used to test the hypothesis. The sequential probability ratio test of a statistical hypothesis is carried out without a predetermined number of observations. Instead, at any stage of the experiment, it makes decision to accept/reject the hypothesis, or to continue the experiment by making an additional observation. The process is terminated when, and only when, the hypothesis is accepted or rejected [11]. Study [13] has conducted such techniques to answer numerical properties of stochastic timed automata.

3 Modeling and Evaluating WSN Protocols with Stochastic Timed Automata

In this section, we present a method to model and evaluate WSN protocols thoroughly:

- For modeling, first, model a protocol's work flow in ideal environment with timed automata. Then, in order to model the uncertainties in WSN behaviors, the work flow timed automata should be extended with stochastic transitions, getting the stochastic timed automata.
- For evaluation, classical model checking can be performed on the ideal work flow timed automata with small scale system to answer whether the design of the system is correct or not. Statistical model checking can be conducted on the stochastic model to evaluate the performance of the system with realistic scale under different environments.

3.1 Modeling of WSN Protocols

3.1.1 Model WSN Protocols with Timed Automata

In most of the WSN protocols, nodes/sensors may bear certain functional roles, i.e., sender and receiver, in the work progress of the protocol. For all the nodes with the same role, they basically share similar behaviors. Thus, we can use a template timed automaton to describe the work flow of these nodes in ideal environment in general. Then, each node can be assigned with a unique number as its identity. Different nodes can communicate with each other through shared variables and synchronize with each other through synchronization messages.

For a template timed automaton which describes the behaviors of a node with certain role, we propose to build the template model in a bottom-up style. By bottom-up, we mean the life cycle of a protocol can be divided into different modules/phases. The fragment of timed automaton for each specific module/phase can be built independently. Then, they can be combined together in the end, which can make the modeling effort much easier and more controllable.

3.1.2 Extend Timed Automata with Stochastic Transitions

WSNs usually work in harsh environments. Thus they may encounter lots of uncertainties such as message loss and node dynamics in reality. To address these aspects and make the model more realistic, we propose to extend the timed automaton with stochastic transitions. The extension is made in two steps. First, add probability factors to the transitions where messages are received to model the message loss. Then node dynamics are described by introducing probability branches to model the scenarios where nodes may fail and resurrect.

Considering of the message loss, when a message is broadcasted, instead of all nodes receiving the message successfully, each node has a probability of failing to receive this message. That is, some of its neighbors can receive this packet successfully, while some of them can not get the packet. To address this aspect, in our model, we add probabilistic factors at the receiver of a message, the probability of receiving it successfully is marked as SUC and the probability of lost this message, is *FAIL*. When the system fires the SUC branch, variables are updated according to the received message and the timed automaton goes to the next state. On the other hand, when the *FAIL* branch is fired, the automaton ignores the message and goes back to the original state. As an example, Fig.1.(a) shows the receiving of *message*.

Since nodes in a WSN network have limited resource and are usually deployed in harsh environmental conditions, WSNs have high dynamics with the constantly changing of network topology. That is, nodes in the network fail and resurrect frequently. To model such behaviors, we introduce a new state, *Failed*, to denote the situation that a node has died and thus will ignore any coming message in the system. Theoretically, a node can fail at any time, i.e. at any state in a timed automaton. We model this scenario by allowing a node enter the *Failed* state from any state randomly. As shown in Fig.1.(b), a node in a regular state can jump to *Failed* state with probability *FAIL*, or stay in this state with probability SUC. Nodes in the *Failed* state can also resurrect and join the network again.

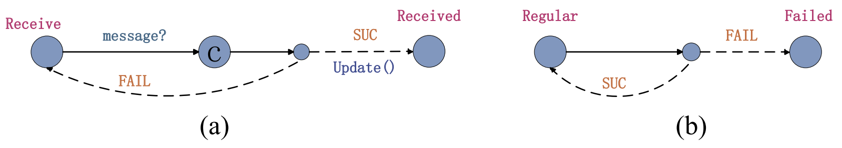


Fig. 1. Model Uncertainties in WSNs

By now, we can get both the ideal and the stochastic timed automata for a WSN protocol. After that, the correctness verification of the design of the protocol can be performed on the classical timed automaton. The performance evaluation of the protocol under different kinds of environments can be conducted on the stochastic model.

3.2 Evaluation of WSN Protocols

3.2.1 Correctness Verification by Classical Model Checking

Correctness verification is conducted on the ideal timed automata by classical model checking techniques. In this phase, we can check functional properties of the protocol, such as whether there are deadlocks, whether all nodes can be eventually synchronized and etc. As classical model checking for timed automata has very high complexity, the verification can only be performed on systems with small scale. By checking functional properties, we can see the correctness of its logic design, and find bugs in the early phase.

3.2.2 Performance Evaluation by Statistical Model Checking

Compared with classical model checking, statistical model checking is much cheaper and has much better scalability. Using the statistical model checking, we can check numerical properties in the stochastic timed automata. For example, we can check the probability that a system can satisfy a given property in bounded time, like: $Pr[time \leq bound](<> expr)$. By collecting and introducing the real probability factors of the link and environment into the model[2], it is easy to evaluate the performance of the candidate protocol in target environment.

Furthermore, when designing or deploying a protocol, there may be many key parameters whose values are crucial for the performance of the complete system. Statistical model checking allows us to check and compare the probabilities that the system can satisfy certain requirement under different candidate values. Thus, by statistical model checking on the stochastic model, we can achieve the target of parameter configuration as well.

4 Modeling and Evaluation of TPSN with Stochastic Timed Automata

To illustrate the feasibility and scalability of the method we presented, in this section, we give the details of modeling and evaluating a time synchronization protocol,

[2] The real values of these factors can be acquired from the network profile.

the Timing-sync Protocol for Sensor Networks (TPSN).

4.1 Modeling of TPSN

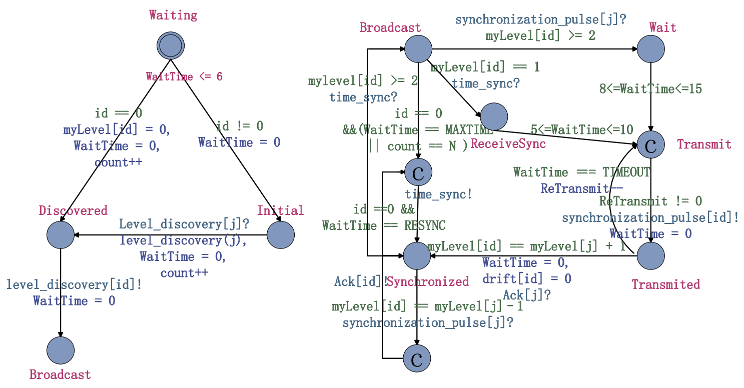
The Timing-sync Protocol for Sensor Networks (TPSN) [4] is a network-wide time synchronization protocol for WSNs. The work flow of TPSN is as follows: a hierarchical structure is created in the network with a designated node as the root, and all the other nodes' level are assigned after. This is called the *level discovery phase*. Then, each node synchronizes with its upper level node through a two way message exchange, and eventually all nodes synchronize with the root node. This is the *synchronization phase*.

4.1.1 Model TPSN with Timed Automata

As mentioned before, in our approach, the timed automata are built in a bottom-up style. Details of timed automaton for each phase are as follows:

Level Discovery Phase: The level discovery phase aims to establish a hierarchical structure in the network, with a designated node as the root and each other node assigned a level. This part of the timed automaton is shown in Fig.2.(a). In our model, system is allowed to leave *Waiting* state in a short period ($WaitTime \leq 6$, in this example). When the network starts, the designated node, i.e., the root node, is assigned level *0*, and goes directly to the *Discovered* state, while other nodes go to the *Initial* state. The root node starts the level discovery phase by broadcasting a *level_discovery* packet and enters the *Broadcast* state, meaning it has done its job in this phase. Neighbors of the root node who receive the *level_discovery* packet go to the *Discovered* state if they are in the *Initial* state, otherwise they will ignore the message. Then nodes in *Discovered* state will broadcast their *level_discovery* messages to neighbors. This procedure will go on until every node in the network is assigned a level and enters the *Broadcast* state.

Synchronization Phase: When the level discovery phase is over, that is, every node in the network is assigned a level ($count == N$, N is the node number of the network) or the root node has stayed in the *Broadcast* for a long enough period of time ($WaitTime == MAXTIME$), the root node starts the synchronization phase by broadcasting a *time_sync* packet. As the root node holds the global standard time, it goes directly to the *Synchronized* state. Nodes in level *1* receive the *time_sync* packet, waiting for a random time ($5 \leq WaitTime \leq 10$), and then go to the *Transmit* state, which means they can start to exchange messages with the root node. By broadcasting a *synchronization_pulse* packet, a node enters *Transmitted* state, waiting for acknowledgements from upper nodes in the *Synchronized* state. On receiving an acknowledgement message *Ack*, the node adjusts its clock to the root node (*drift[id]==0*). All these messages are broadcasted and a node in level *2* has at least one neighbor in level *1*, so it can overhear these message exchanges. When a node receives a *synchronization_pulse* from upper level nodes, it will back off for a random time ($8 \leq WaitTime \leq 15$), waiting for the end of the message exchange between the upper level nodes. Then it initiates message exchange with upper level nodes. This process is carried out throughout the network until all



(a) Level Discovery Phase (b) Synchronization Phase

Fig. 2. Timed Automata of TPSN

nodes enter the *Synchronized* state.

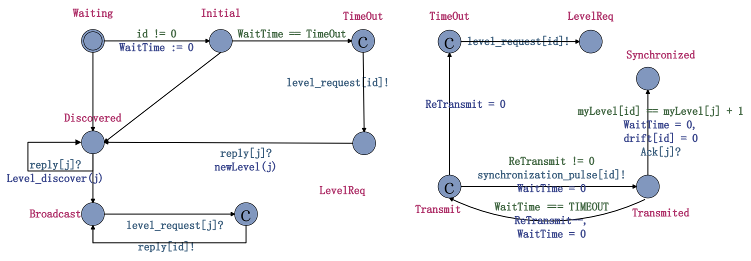
Due to clock drift among sensor nodes, it is necessary to resynchronize periodically to limit the drifts between all nodes in a small range. The resynchronization phase is almost same with the synchronization phase. The root node broadcasts the *time_sync* packet again after it staying in the *Synchronized* state for a fixed period. Other nodes in the *Synchronized* state will wait for some time then broadcast *synchronization_pulse* packets to start message exchange with the root node if they are in level *1*. Otherwise, they will jump to the *Broadcast* state to wait for *synchronization_pulse* from upper level. The timed automaton of synchronization phase is depicted in Fig.2.(b).

Special Provisions: In our timed automata, we also model the special cases that the authors emphasize in the profile [4].

First, local level discovery is caused by new nodes joining to the system after the main part of the network have already finished level discovery phase. As shown in Fig.3.(a), if a node has been in *Initial* state for a period but has not receive any *level_discovery* message from neighbor nodes, it timeouts and broadcasts a *level_request* message to request a level. The neighbors reply to this request by sending their levels. If there are two or more replies, it assigns itself a level one greater than the smallest one. As the time when a node can join the network is unlimited, we make a node capable of replying the *level_request* packet at any time as long as it has been assigned a level.

Second, sensor nodes may also die randomly. A node would retransmit the *synchronization_pulse* after a period of time if it cannot get an acknowledgement back. After retransmitting the *synchronization_pulse* a fixed number of times, a node assumes that it has lost all its neighbors in the upper level and jumps to the *TimeOut* state, then it broadcasts a *level_request* message. On getting back a reply, the node is assigned a new level and joins the hierarchical structure again. This part of timed automaton is shown in Fig.3.(b).

So far, we can combine all parts of the timed automaton together and get a complete model to describe the whole work flow of TPSN in ideal environment, as shown in Fig.4.



(a) Local Level Discovery Phase (b) Level Re-discovery Phase

Fig. 3. Timed Automata of TPSN

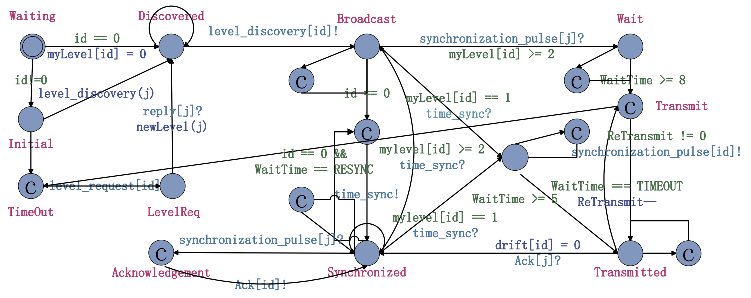


Fig. 4. Complete Ideal Timed Automaton of TPSN

4.1.2 Extend Timed Automata of TPSN with Stochastic Transitions

In wireless networks, communication is unstable and message collision is inevitable when messages are broadcasted. As a result, message loss is common. As proposed in section 3, we use stochastic transitions to address such phenomenons in our timed automata. As an example, Fig.5.(a) shows the receiving of *level_discovery[id]*. To model the phenomenon that any message may loss in a sensor network, we add probabilistic extension to every message receiving transitions in the model.

Similarly, to model node dynamics, we introduce the *failed* state denoting that a node is dead and will ignore any messages. We allow an automaton to enter the *failed* state randomly from any states. As shown in Fig.5.(b), a node in a regular state can jump to *Failed* state with probability *FAIL*, or stay in this state with probability *SUC*. Nodes in the *Failed* state can also resurrect and join the network again. If the node is not the root node, it goes to the *Waiting* state with probability *SUC* and waits to be assigned a new level. Note that, as the root node is designated, the dynamic of the root node is simplified in the model. The root node may enter the *Failed* state from any other states, similar with regular nodes. But when it is in the *Failed* state, it resurrects and enters the *Waiting* state directly, then restarts the timing-sync protocol in the whole network.

As shown in Fig.6, by integrating the above methods into the ideal model, the uncertainties in the real-time behavior of WSN protocols can be described in the model now. Due to space limitation, readers are referred to link `http://seg.nju.edu.cn/people/~bl/exp/TPSN.rar` for the complete models.

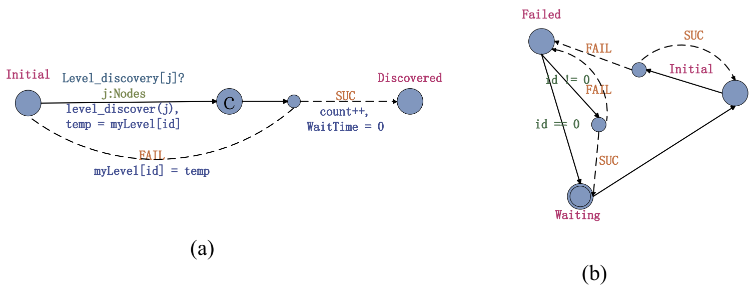


Fig. 5. Extended Timed Automaton of TPSN with Probability

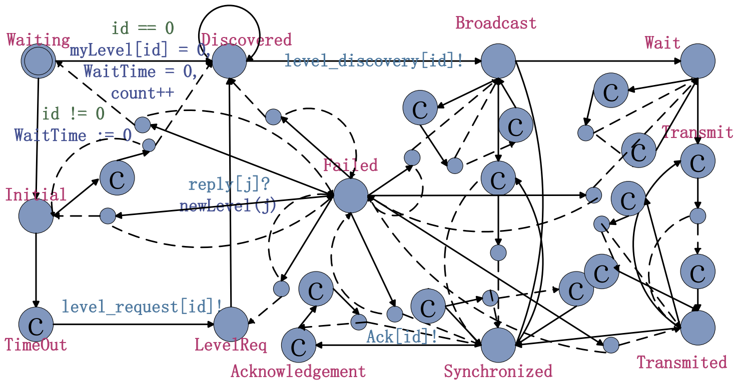


Fig. 6. Stochastic Timed Automaton of TPSN

4.2 Verification of TPSN

In section 4.1, we give both the ideal and stochastic timed automata of TPSN respectively. In this section, we will analysis TPSN based on these models. We verify the ideal timed automata against typical functional properties and evaluate the performance of TPSN under different environments using statistical model checking technique on the stochastic timed automata.

The system we verify is a TPSN system with N nodes. In another word, it is a timed automata network consists of N automata according to the models built in the last sections. The topology of the TPSN system studied here is with full connectivity. All the nodes in the network are neighbors with each other[3]. The computer we use is configured with Intel(R) Core(TM) 2 Quad Q9500 processor, 2G RAM and Windows 7 Professional. The model checker we use is UPPAAL (V.4.1.7) [7]. The false negatives (α) and false positives (β) used in the statistical model checking experiments are both set as 0.05.

4.2.1 Correctness Verification of TPSN

TPSN aims to achieve network-wide synchronization. In the protocol profile, the process is specified in two phases, so the correctness of these two phases needs to be checked respectively. The properties we checked are as follows:

[3] The topology of the system under verification can be easily changed by modifying the topology matrix in the model. The readers are referred to the complete models for detail.

Table 1
Correctness Verification Results in the Timed Automata

N	**3**	**4**	**5**	**6**
Level Assignment(C(T/M))	Y(0.031/13)	Y(0.032/19)	Y(0.25/32)	Y(3.791/17)
Node Synchronization(C(T/M))	Y(0.031/14)	Y(0.53/20)	Y(43.384/31)	Y(16984.546/47)
Bounded Time Range(C(T/M))	Y(0.031/15)	Y(0.249/19)	Y(10.28/47)	Y(3024.326/430)
Deadlock Free(C(T/M))	Y(0.047/15)	Y(0.375/18)	Y(15.865/47)	Y(3603.202/580)

- **Level Assignment:** Check whether every node in the network can be assigned a level. In this model, a node's level is initiated with N, and once it is assigned a level, the level value is less than N. So, this property can be specified by a CTL formula as: $A <> \forall(id : Nodes) myLevel[id] < N$.
- **Node Synchronization:** Check whether every node in the network can eventually enter the *Synchronized* state. This is expressed by: $A <> \forall(id : Nodes) tpsn(id).Synchronized$.
- **Bounded Time Range:** Another property to see whether TPSN can work well is to check whether the clock drift between regular nodes and the root node is bounded in a reasonable range. This property is checked by: $A[] drift[id] < Num$.
- **Deadlock Free:** Last but not the least, the absence of deadlock: $A[]$*not deadlock*.

We check these properties on networks with 3-6 processors/nodes in the timed automata with UPPAAL. All these properties are satisfied, which means the TPSN can work correctly in the ideal environment. The verification results (whether the property is satisfied (C:Y/N), CPU time (T:Second) and memory usage (M:MB)) are shown in Table 1.

The verification results show that the design logic of TPSN is correct basically, i.e., it can work well if all messages are transmitted without any error and nodes work well all the time.

4.2.2 Performance Measurement of TPSN

Except to know whether the functionality of the protocol is correct or not, when designing a protocol, or choosing which protocol to use in a system, the designers/users are always interested in the performance of the protocol under certain target environment. Actually, in TPSN, there are many factors which are closely related with the system's performance as:

- **Resynchronization Interval:** Resynchronization Interval ($RESYNC$) is an important factor in the design of TPSN. The value of this factor has a great effect on the system's efficiency. There is a tradeoff when choosing the value of $RESYNC$. It should be long enough to help the synchronization complete in a synchronization cycle; On the other hand, it also has to be as short as possible to limit the clock drift between two nodes in a small range [4]. Thus, it is helpful to find a way to make the choice of a proper value for this factor easier.
- **Synchronization Time:** When TPSN works well, one important aspect people care is the efficiency of the protocol, like how long it takes to synchronize all the nodes. To analyze the efficiency, we verify the probabilities of all nodes

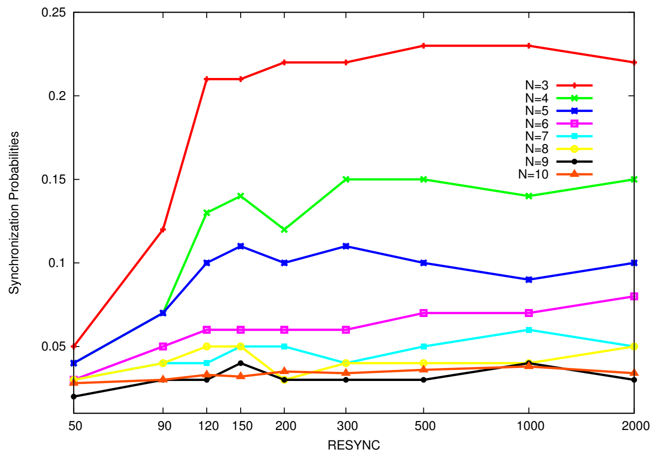


Fig. 7. Data of $RESYNC$ with Failure Probability = 0.1

be synchronized successfully in M resynchronization phases, i.e., $Pr[time <= M * RESYNC](<> forall(i : Nodes)tpsn(i).Synchronized)$

- **Failure Probability:** Failure Probability is the value of the probability of message loss and/or node dynamic in the system. Its value is decided by $FAIL/(FAIL + SUC)$ in our model. Users would like to know how it can impact the performance of TSPN. For example, if the system is suppose to work in a terrible environment, then if the influence of failure probability is huge, the protocol may not be a suitable choice.

Now, we start to analyze the performance of TPSN numerically in the stochastic timed automata with respect to these key factors by statistical model checking as follows.

Resynchronization Interval: We have modeled resynchronization in our timed automata. To show how $RESYNC$ affects the performance of TPSN, we verify a series of networks with 3-10 nodes and record the probabilities that a system can get synchronized in a cycle with different $RESYNC$. The verification results are plotted in Fig.7, with failure probability as 0.1.

From this figure, we can see that with the increasing of *RESYNC*, the probability of synchronizing all nodes in a single cycle increases rapidly at first; then when *RESYNC* value gets close to 150, the probability trends to be stable; once *RESYNC* is bigger than 150, the probability only fluctuates slightly.

We can see that even when $RESYNC$ is large enough, i.e, 2000, the probability of all nodes to get synchronized in the first cycle is approximate 0.2 in the best case. The reason is that nodes could be timeout repeatedly and thus can not be synchronized with others in one cycle in the extreme case. Therefore, instead of increasing the value of $RESYNC$ and waiting for all nodes to get synchronized, we can choose a proper value of $RESYNC$, then the system can start over again quickly. From this study, 150 time units seem to be a suitable reference value for $RESYNC$. Therefore, in the following experiments $RESYNC$ will be set as 150.

Table 2
Synchronization Probabilities in M Intervals in Stochastic Timed Automata with Failure Probability set 0 and 0.1

	Failure Probability = 0			**Failure Probability = 0.1**			
N	**M=1**	**M=2**	**M=3**	**M=1**	**M=10**	**M=100**	**M=400**
3	[0.84,0.94]	[0.95,1]	[0.95,1]	[0.17,0.27]	[0.31,0.41]	[0.65,0.75]	[0.93,1]
4	[0.86,0.96]	[0.95,1]	[0.95,1]	[0.08,0.18]	[0.14,0.24]	[0.40,0.50]	[0.82,0.92]
5	[0.88,0.98]	[0.95,1]	[0.95,1]	[0.02,0.12]	[0.09,0.19]	[0.27,0.37]	[0.62,0.72]
6	[0.88,0.98]	[0.95,1]	[0.95,1]	[0.03,0.13]	[0.03,0.13]	[0.13,0.23]	[0.41,0.51]
7	[0.89,0.99]	[0.95,1]	[0.95,1]	[0,0.10]	[0.02,0.12]	[0.07,0.17]	[0.25,0.35]
8	[0.89,0.99]	[0.95,1]	[0.95,1]	[0,0.07]	[0,0.09]	[0.01,0.11]	[0.15,0.25]
9	[0.87,0.97]	[0.95,1]	[0.95,1]	[0,0.06]	[0,0.07]	[0,0.09]	[0.06,0.16]
10	[0.88,0.98]	[0.95,1]	[0.95,1]	[0,0.06]	[0,0.07]	[0,0.07]	[0.02,0.12]
20	[0.85,0.95]	[0.95,1]	[0.95,1]	[0,0.05]	[0,0.05]	[0,0.05]	[0,0.05]

Synchronization Time: When the TPSN works well and all nodes in the network can be synchronized eventually, we are interested in the time it takes to synchronize all the nodes to evaluate the efficiency of the protocol.

The verification results of probabilities of all nodes get synchronized in M cycles in stochastic model with failure probability set as 0 and 0.1 are shown in Table 2. We can see that when the environment is stable, with the increase of node number in the network, the probability of successfully synchronizing in certain periods becomes smaller, which means it takes longer to get synchronized. Furthermore, when the environment getting worse, the performance of TPSN drops dramatically. For example, when failure probability is 0, for a system with 10 nodes, 3 cycles are enough for all the nodes to get synchronized. When failure probability is 0.1, for the same problem, even 400 cycles, the probability that all the nodes are synchronized is only around [2%, 12%]. Furthermore, even there are only 3 nodes in a network, it takes more than 400 cycles to make sure the network is synchronized. We increase the number of the nodes in the system continuously and find that when the number of nodes reaches 20, the probability to make the whole system get synchronized is stable in less than 5%. The same story happened on the system with 100 nodes. **These phenomenons means for a TPSN system with moderate size, e.g., more than 20 nodes, working in a environment with even slight disturbance, TPSN is nearly impossible to work correctly.** Therefore, we only report the data with 3-20 nodes in this study. From these data, we are surprisingly to see that the performance of TPSN is sensitive to environment. Furthermore, the disturbance caused by failure probability is more severe for system with more nodes. Thus, we decide to dig deeper to see how can the environment can affect TPSN actually.

Failure Probability: In the last experiments, we find that message loss and node dynamics could affect the performance of TPSN a lot. To know the details of how the environment can impact TPSN, for example what the performance of the protocol will be if the environment is getting worse, more experiments are conducted. We check the probabilities of a system consisting of 3 nodes to synchronize successfully in M cycles with failure probability set as 0, 0.1, 0.2 and 0.3 respectively. The results are plotted in Fig.8.

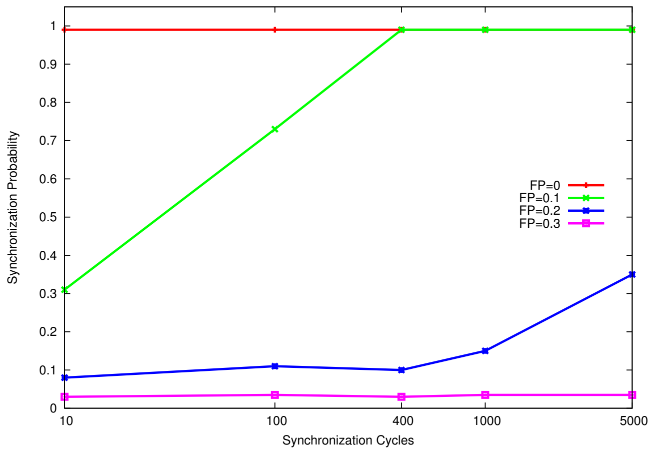


Fig. 8. Synchronization Probabilities with Different Failure Probability(FP)

Data in Fig.8 show that with the increasing of failure probability, the performance of TPSN drops extremely quickly. When failure probability is 0, i.e, the protocol works in an ideal environment, we can almost guarantee that all nodes can get synchronized in less than 10 resynchronization periods. When failure probability is 0.1, it takes longer for all nodes to be synchronized. However, it can still be guaranteed that all the nodes can be synchronized in 400 cycles. Meanwhile, when failure probability is 0.2, the probability is less than 40% in even 5000 cycles. Not to mention when failure probability is 0.3, in the same system, it is almost impossible to synchronize all nodes in 5000 cycles. **From these studies, we can tell that the performance of TPSN is very sensitive to message loss and node dynamics, even a small WSN system, with only 3 nodes, could not perform well in harsh environments**.

5 Conclusion

In this paper, we propose a complete method to analyze and evaluate WSN protocols. By our method, we can model not only the ideal work flow of a WSN protocol, but also the nondeterministic behaviors which are quite common in WSN systems' behavior, like message loss and node dynamics. Furthermore, besides of checking the correctness of the protocol, we propose to use statistical model checking to evaluate the performance of the protocol numerically as well.

To illustrate the feasibility and scalability of our approach, a real-case WSN protocol TPSN is studied thoroughly in this paper. It indicates that stochastic timed automata and statistical model checking are very helpful in analyzing large WSN systems working in unreliable environments. Furthermore, as statistical model checking is much easier to conduct and less expensive than classical model checking, it is possible to examine real-case large systems, e.g. a system consists of 100 nodes by statistical model checking, while classical model checking can only able to handle system up to 6 nodes. We show that it is possible and very convenient to use statistical model checking to analyze large WSN systems numerically for many different objectives, e.g., parameter configuration, performance evaluation and even candidate protocol selection.

Acknowledgement

We would like to thank the anonymous reviewers for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (No.61100036, No.61170066, No.61003025), the National Grand Fundamental Research 973 Program of China (No.2009CB320702), the National 863 High-Tech Programme of China (No.2011AA010103, No.2012AA0
11205), the Jiangsu Province Research Foundation (No.BK2011558, BK2010170), and by the Research Fund for the Doctoral Program of Higher Education of China (No.20110091120058).

References

[1] Yick, J., B. Mukherjee, and D. Ghosal, *Wireless Sensor Network Survey.* Computer Networks 52. 2292-2330, 2008

[2] Dwyer, M. B., J. Hatcliff, Robby, C. S. Păsăreanu, and W. Visser, "Formal Software Analysis Emerging Trends in Software Model Checking". FOSE. 120-136, 2007

[3] Basu, A., S. Bensalem, M. Bozga, B. Delahaye and A. Legay, *Statistical abstraction and model-checking of large heterogeneous. International Journal on Software Tools for Technology Transfer(STTT).* Volume14, Number 1 (2012), 53-72, 2012

[4] Ganeriwal, S., R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks". In Proceedings of the SenSys'03, 138-149. ACM, 2003

[5] Alure, R., and D. L. Dill, "A Theory of Timed Automata." TCS, 126(2), 1994

[6] Maróti, M., B. Kusy, G. Ssimon, and Á. Lédeczi, "The Flooding Time Synchronization Protocol". In Proceedings of the SenSys'04. 39-49. ACM, 2004

[7] UPPAAL, http://www.uppaal.com/

[8] Younes, H. L. S., and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling". In CAV, LNCS 2404, 223-235. Springer, 2002

[9] Bengtsson, J., and Y. Wang, "Timed Automata: Semantics, Algorithms and Toos". In Lectures on Concurrency and Petri Nets, LNCS 3098, 87-124, 2004

[10] Younes, H. L. S., M. Kwiatkowska, G. Norman, and D. Parker, *Numerical vs. Statistical Probabilistic Model Checking.* International Joural on Software Tools for Technology Transfer (STTT), Volume 8, Number 3(2006), 216-228.

[11] Wald, A., *Sequential tests of statistical hypotheses.* Annals of Mathematical Statistics, 16(2):117C186, 1945.

[12] Younes, H. L. S., "Verification and Planning for Stochastic Processes with Asynchronous Events." PhD thesis, Carnegie Mellon, 2005

[13] David, A., K. G. Larsen, A. Legay, M. Mikučionis, and Z. Wang, "Time for Statistical Model Checking of Real-Time Systems". CAV, LNCS 6806, 349-355, 2011

[14] Kusy, B. and S. Abdelwahed, "FTSP protocol verification using SPIN." Technical Report ISIS-06-704, Institute for Software Integrated Systems, 2006

[15] McInnes, A. I., "Model-checking the Flooding Time Synchronization Protocol". In Proceedings of the 7th IEEE-ICCA, 2009

[16] Tan, L., L. Bu, J. Zhao and L. Wang, "Analyziang FTPS Robustness With Timed Automata". The Second Asia-Pacific Symposium on Internetware, 2010

[17] Heidarian, F., J. Schmaltz, and F. Vaandrager, "Analysis of a Clock Synchronization Protocol for Wireless Sensor Networks". FM2009, LNCS 5850, pp. 516-531, 2009

[18] Schuts, M., F. Zhu, F, Heidarian, F, Vaandrager, "Modelling Clock Synchronization in the Chess gMAC WSN Protocol". Workshop on Quantitative Formal Methods: Theory and Applications EPTCS, pp. 1-14, 2009

[19] Huang, X., A. Singh, S. A. Smolka, "Using Integer Clocks to Verify the Timing-Sync Sensor Networks Protocol". Proceedings of NFM, 2010

[20] Abo, R. and K. Barkaoui, *A Performability Analysis of Mobile Wireless Sensor Networks with Probabilistic Model Checking*. Wireless Advanced, 283-288, 2011

[21] PRISM, http://www.prismmodelchecker.org/

[22] Zhang, F., L. Bu, L. Wang, J. Zhao, X. Li., "Poster Abstract: Numerical Analysis of WSN Protocol Using Probabilistic Timed Automata". In the Proceedings of ICCPS 2012.