Theoretical Computer Science 458 (2012) 61–75

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A state-based model of sensor protocols☆

Young-ri Choi [a,*], Mohamed G. Gouda [b]

[a] School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology, Ulsan, 689-798, Republic of Korea
[b] Department of Computer Science, The University of Texas at Austin, Austin, TX, 78701, USA

ARTICLE INFO

Article history:
Received 29 April 2011
Received in revised form 15 August 2012
Accepted 20 August 2012
Communicated by M. Mavronicolas

Keywords:
Sensor network
State-based model
Formal model
Protocol specification
Protocol verification and analysis
Self-stabilization

ABSTRACT

We introduce a state-based model that can be used in specifying and verifying the desired properties of sensor network protocols. This model accommodates several characteristics that are common in sensor networks. Examples of these characteristics are unavoidable local broadcast, probabilistic message transmission, asymmetric communication, message collision, and timeout actions and randomization steps. We also propose three methods for analyzing a sensor network protocol specified in the model. In the first method, called *nondeterministic analysis*, a specified sensor network protocol is shown to be "nondeterministically correct" under the assumption that message delivery is assured and message collision is guaranteed not to occur. In the second method, called *probabilistic analysis*, the protocol is shown to be "probabilistically correct" under the assumption that message delivery is probabilistic but message collision is guaranteed not to occur. In the third method, called *collision analysis*, the effect of message collision and probabilistic message delivery on the correctness of the protocol is analyzed. To demonstrate the utility of our model, we discuss an example protocol that can be used by a sensor to identify its strong neighbors in the network, and apply the analysis methods to the protocol. Using our model, we also show that the protocol satisfies a desirable property, called self-stabilization property.

1. Introduction

A sensor is a battery-operated small computer with an antenna and a sensing board that can sense magnetism, sound, heat, etc. Sensors in a network can use their antennas to communicate in a wireless fashion by broadcasting messages over radio frequency to neighboring sensors in the network. Due to the limited range of radio transmission, sensor networks are usually multi-hop. Sensor networks can be used for military, environmental or commercial applications such as intrusion detection [2], disaster monitoring [3] and habitat monitoring [4].

Sensor networks and their protocols have several characteristics that make them hard to specify formally and even harder to verify. Examples of these characteristics are

i. *Unavoidable local broadcast:* When a sensor sends a message, even one that is intended for a particular neighboring sensor, a copy of the message is received by every neighboring sensor.
ii. *Probabilistic message transmission:* When a sensor sends a message, the message reaches the different neighboring sensors (and can be received by each of them) with different probabilities.

☆ This work was partly supported by the Defense Advanced Research Projects Agency (DARPA) Contract F33615-01-C-1901. A preliminary version of this paper appeared in the 9th International Conference on Principles of Distributed Systems (OPODIS 2005) [1].

* Corresponding author. Tel.: +82 52 217 2136; fax: +82 52 217 2130.
E-mail addresses: ychoi@unist.ac.kr (Y.-r. Choi), gouda@cs.utexas.edu (M.G. Gouda).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.08.013

iii. *Asymmetric communication:* Let u and v be two neighboring sensors in a network. The probability that a message sent by u is received by v can be different from the probability that a message sent by v is received by u.
iv. *Message collision:* If two neighboring sensors send messages at the same time, then neither sensor receives the message from the other sensor. Moreover, if two (not necessarily neighboring) sensors send messages at the same time, then any sensor that is a neighbor of both sensors will not receive any of the two messages. In this case, the two messages are said to have collided.
v. *Timeout actions and randomization steps:* Given the above characteristics of a sensor network, it seems logical that sensor network protocols need to heavily depend on timeout actions and randomization steps to perform their functions.

The above characteristics of sensor network protocols (for short, sensor protocols) are far from common in the literature of distributed systems. Thus, one is inclined to believe that the "standard model" of distributed systems is not suitable for sensor protocols. The search for a suitable model for sensor protocols is an obligatory first step toward formal specification, verification, and design of these protocols.

In this paper, we present a state-based model of sensor protocols, which accommodates the above characteristics of sensor networks. This model can be used in specifying and verifying the desired properties of sensor protocols. The presented model of sensor protocols is rather detailed. Thus, we also propose three methods for analyzing sensor protocols specified in the model. In the first method, called *nondeterministic analysis*, a specified sensor protocol is shown to be "nondeterministically correct" under the assumption that message delivery is assured and message collision is guaranteed not to occur. In the second method, called *probabilistic analysis*, the protocol is shown to be "probabilistically correct" under the assumption that message delivery is probabilistic but message collision is guaranteed not to occur. In the third method, called *collision analysis*, the effect of message collision and probabilistic message delivery on the correctness of the protocol is analyzed, and appropriate values for some parameters in the protocol are chosen to achieve a compromise between two conflicting factors: make the probability of message collision reasonably small, and make the protocol reach a target state within a reasonably short time. The proposed analysis methods offer sensor protocol designers and developers clear insights on how a sensor protocol works under different assumptions about message transmission and collision.

To demonstrate the utility of the model, we discuss, as an example, a neighbor computation protocol that can be used by a sensor to identify its strong neighbors in the network. We formally specify the protocol using the model, and then apply the three analysis methods to the protocol. Also as a property verification example, we prove that the protocol satisfies a desirable property, called self-stabilization property.

The proposed analysis methods analyze a sensor protocol under different assumptions about message transmission and collision. Under the different assumptions, we use different approaches. For the nondeterministic analysis, we generate a state transition diagram of the protocol to verify that it is guaranteed to reach a target state. For the probabilistic analysis, we generate a probabilistic state transition diagram of the protocol to verify that it reaches a target state with a high probability. At last, for the collision analysis, we simulate the protocol and analyze the effect of collision on the execution of the protocol.

The rest of the paper is organized as follows. In Section 2, we discuss related work and motivation of this work. In Sections 3 and 4, we present a topology and execution model of sensor networks. In Section 5, we discuss three analysis methods of sensor protocols. We present a neighbor computation protocol in Section 6, apply the three analysis methods to the protocol in Section 7, and prove that the protocol is self-stabilizing in Section 8. We discuss issues on our analysis process and our model for complex sensor protocols in Section 9, and finally make concluding remarks in Section 10.

2. Related work and motivation

A formal model of sensor protocols allows sensor protocol developers to formally specify and verify the desired properties of sensor protocols. Also, it allows the developers to develop the simulation of the protocols based on the model, so that they can evaluate the performance of the protocols. Moreover, in the design stage of sensor protocols, the developers can use a formal model to explore the implications of design decisions on the protocols, and performance trade-offs between the protocols. Simulation and testing have been used to verify sensor protocols as well as other network protocols, but they cannot guarantee the desirable properties or correctness of the protocols. By employing a formal model along with simulation and testing, the developers can decrease the development time, and improve the reliability of the protocols [5–8].

The unique characteristics of sensor networks described in the previous section make sensor protocols hard to specify formally and even harder to verify or analyze. Moreover, these characteristics distinguish sensor networks from other distributed systems, and make existing models for distributed systems unsuitable for sensor networks. Thus, we need to investigate a suitable model of sensor protocols, which accommodates these characteristics.

Several sensor protocols have been developed based on abstract models. In these models, each sensor communicates with other sensors using shared variables [9], abstract channels [10], and FIFO message channels that can hold many messages [11]. Also in [12], it is assumed that communications between neighboring sensors are reliable. We believe that these models are not easily realized in sensor networks, and thus they are not suitable for designing and analyzing sensor protocols. There have been some efforts on transformations of protocols from the models of distributed systems to a sensor network model [13–15]. However, these models assume symmetric communications between sensors, and do not consider probabilistic message transmission, and timeout actions and randomization steps, which are very common in practical sensor networks. Preliminary study on a suitable model of sensor protocols was discussed in [1].

Several models for sensor applications have been proposed [16–20]. In general, the purpose of these models is to hide application programmers from low-level details such as routing, group management, resource management, etc. A high-level programming abstraction was proposed to write a tracking application in sensor networks without knowing low-level details [16]. In [19], a state-centric programming model was investigated to write sensor applications as the computation of state updates. In [17], a functional language was introduced to specify the global behavior of a sensor application. In addition, database approaches were proposed in [18,20], where a high-level query language is used to describe sensor applications.

It is important to state here that all the above efforts are directed toward modeling sensor network applications. Clearly, sensor protocols are quite different from sensor applications in terms of their functions and in terms of how they accomplish these functions. For instance, sensor protocols need to deal with the intricate characteristics of sensor networks, as they attempt to hide these characteristics from the sensor applications. Thus, whereas a sensor protocol has to deal with unavoidable local broadcast, probabilistic message transmission, message collision, etc., a sensor application can view the sensor network as a reliable medium for communicating sensing data. Also whereas a sensor protocol depends heavily on timeout actions and randomization steps, a sensor application rarely needs to resort to these devices.

Formal verification of routing protocols in wireless networks has been studied, using various verification systems or tools of modeling distributed systems. In [7], Câmara et al. discussed existing formal verification tools of distributed systems for modeling wireless network protocols. The AODV routing protocol was analyzed using Colored Petri Nets [8], and the model checker UPPAAL [21]. In [22], two automatic model checking tools, SPIN and UPPAAL, were used to verify the Lightweight Underlay Network Ad hoc Routing protocol. In [6], a technique was proposed to formally verify routing protocols without modeling a particular network configuration. In addition, it has been studied to derive finite state machines from programs [23–25], and to automatically generate code for multiple platforms (like TinyOS [26], and MANTIS [27]) [28,29]. Kothari et al. proposed a program analysis tool that generates finite state machines of TinyOS components automatically for error detection and program validation [23]. Their work has a limitation on capturing some state transitions by timeout events, on which sensor protocols heavily depend.

Unlike the above work, we focus on investigating what a model should be to describe sensor protocols, and how sensor protocols can be verified or analyzed in the model. Studies on automating our analysis processes such as generating state transition diagrams and simulation codes automatically from a protocol specification, and integrating our model into existing verification systems or tools are beyond the scope of this work. We consider them as our future work.

In [30], formal models, which are specific to flooding and gossiping protocols, were described using PRISM, and the effects of different assumptions on these protocols were investigated. Our proposed model and analysis methods aim to be used for a general class of sensor protocols.

3. Topology of sensor networks

The *topology* of a sensor network is a directed graph where each node represents a distinct sensor in the network and where each directed edge is labeled with some probability. A directed edge (u,v), from a sensor u to a sensor v, that is labeled with probability p, where $p > 0$, indicates that if sensor u sends a message, then this message arrives at sensor v with probability p (provided that neither sensor v nor any "neighboring sensor" of v sends another message at the same time).

There are two probabilities, α and β, where $\alpha > \beta$, that label the edges in the topology of a sensor network. An edge that is labeled with the large probability α is called a *strong edge*, and an edge that is labeled with the small probability β is called a *weak edge*. In this paper, we assign α 0.95 and β 0.5. Below we discuss some experiments that we have carried out on sensors and led us to this choice of probabilities in the topology of a sensor network [31]. It is important to note that although the probability labels of α and β are chosen to be 0.95 and 0.5 in this paper, different values can be chosen for these probability labels for different settings of sensor networks.

Let u and v be two distinct sensors in a network. Sensors u and v are called *strong neighbors* iff there are two strong edges between them in the network topology. The two sensors are called *middle neighbors* iff there are one strong edge and one weak edge between them in the network topology. Sensors u and v are called *weak neighbors* iff there is exactly one edge between them, or there are two weak edges between them in the network topology. They are called *non-neighbors* iff there are no edges between them in the network topology. If there is an edge from u to v in the network topology, then u is called an *in-neighbor* of v and v is called an *out-neighbor* of u.

As an example, Fig. 1 shows the topology of a sensor network that has four sensors. In this network, sensors u and v are weak neighbors, sensors u and v' are strong neighbors, sensors u and v'' are middle neighbors, and sensors v and v'' are non-neighbors. Sensor u has three out-neighbors, namely sensors v, v', and v''. Also sensor u has two in-neighbors, namely sensors v', and v''.

In [31], we describe some experiments that we have carried out using Mica sensors [32]. In these experiments, a sensor u sends a sequence of messages at the rate of one message per 5 seconds, and another sensor v attempts to receive all the sent messages. The results of these experiments are summarized in Fig. 2 where each point represents the result of one experiment. (Similar results are also reported in [33–35].)

We observe that from Fig. 2 if the distance between two sensors u and v is in the range $0 \cdots 38$ inches, v receives between 90% and 100% of the messages sent by u. On the other hand, if the distance between sensors u and v is in the range $38 \cdots 67$ inches, v receives anywhere between 0% and 100% of the messages sent by u. Finally, if the distance between sensors u and

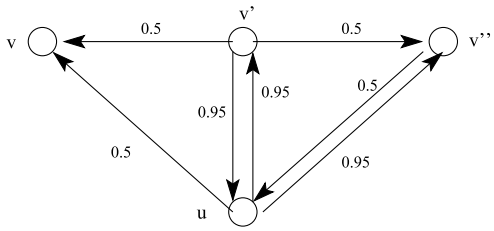

Fig. 1. Topology of a sensor network.

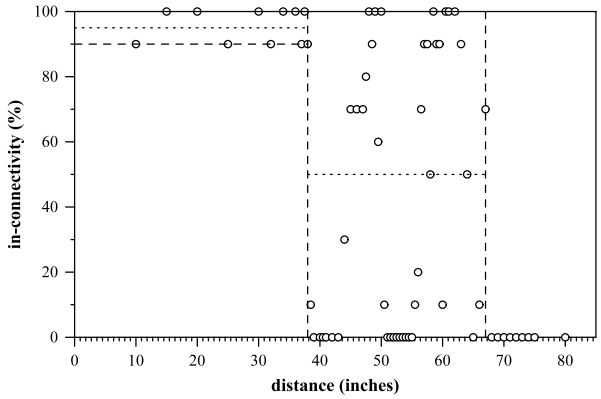

Fig. 2. Percentage of received messages.

```
sensor <sensor name>

const <const name> : <const type>, ... , <const name> : <const type>
var   <var name> : <var type>, ... , <var name> : <var type>

begin
   timeout-expires  -> <action statements>       // timeout action
[] rcv <msg.0>      -> <action statements>       // receiving action
   ...                 ...
[] rcv <msg.k-1>    -> <action statements>       // receiving action
end
```

Fig. 3. Specification of a sensor

v is longer than 67 inches, v receives 0% of the messages sent by u. From these observations, we idealize the diagram in Fig. 2 as follows. If their distance is in the range $0 \cdots x$, then the directed edge from u to v can be labeled with probability 0.95, and the edge is strong. If their distance is in the range $x \cdots y$, then the directed edge from u to v can be labeled with probability 0.5, and the edge is weak.

In ad-hoc and sensor networks, Unit Disk Graph and Quasi Unit Disk Graph models have been used [36–38]. In both models, a network is represented by an undirected graph, where the Euclidean distance between two nodes determines the existence of an edge between them, and an edge between them indicates that they can always communicate with each other directly. Unlike our model, these models do not consider probabilistic message delivery, and asymmetric communication, which are very common in sensor networks.

4. Sensor network execution

A sensor is specified as a program that has global constants, local variables, and one or more actions. In general, a sensor is specified as in Fig. 3. Note that the actions of a sensor consist of exactly one timeout action and zero or more receiving actions. Before we can discuss the execution of sensor actions, we need to explain our model of real-time.

We assume that the real-time passes through discrete time instants: instant 1, instant 2, instant 3, and so on. The time periods between consecutive instants are equal. Executions of the different actions of a sensor occur only at the time instants, and not during the time periods between instants. We refer to the time period between two consecutive instants t and $t + 1$ as the *time unit* $(t, t + 1)$. (The value of a time unit is not critical to the current presentation, but we estimate that the value of the time unit is around 100 ms.)

At a time instant t, if the timeout of a sensor u expires, then u executes its timeout action. Executing the timeout action of sensor u at t causes u to update its local variables, and to send at most one message at t. It also causes u to execute the statement "timeout-after <expression>" which causes the timeout of u to expire (again) after k time units, where k is the value of <expression> at the time unit $(t, t+1)$. The timeout action of sensor u is of the following form:

```
timeout-expires ->
   <update local variables of u>;
   <send at most one message>;
   <execute timeout-after <expression>>
```

To keep track of its timeout, each sensor u has an implicit variable named "timer.u". In each time unit between two consecutive instants, timer.u has a fixed positive integer value. The value of timer.u is determined by the following two rules:

i. If the value of timer.u is k, where $k > 1$, in a time unit $(t-1, t)$, then the value of timer.u is $k-1$ in the time unit $(t, t+1)$.
ii. If the value of timer.u is 1 in a time unit $(t-1, t)$, then sensor u executes its timeout action at instant t. Moreover, since sensor u executes the statement "timeout-after <expression>" as part of executing its timeout action, in the time unit $(t, t+1)$, the value of timer.u is the value of <expression>.

If a sensor u executes its timeout action and sends a message at an instant t, then an out-neighbor v of u receives a copy of the message at the same time instant t, provided that the following three conditions hold.

i. A random integer number is uniformly selected in the range $0 \ldots 99$, and this selected number is less than $100 * p$, where p is the probability label of edge (u, v) in the network topology.
ii. Sensor v does not send any message at instant t.
iii. For each in-neighbor w of v, other than u, if w sends a message at t, then a random integer number is uniformly selected in the range $0 \ldots 99$, and this selected number is at least $100 * p'$, where p' is the probability label of edge (w, v) in the network topology.

The first condition is to make v receive any message sent by u with probability p provided that the other two conditions hold. The second and third conditions are to ensure that the message sent by u does not collide with other messages. If sensor v sends a message at the same instant t, the message sent by v *collides* with the message sent by u. Also if two in-neighbors u and w of v send messages at the same instant t, and selected random numbers for u and w are less than $100 * p$ and $100 * p'$, respectively, then the messages sent by u and w collide with one other. In both cases, v ends up receiving no message at t. Sending operations and their corresponding receiving operations are executed synchronously in the network, and a sensor cannot send and receive messages at the same time instant.

If a sensor u receives a message <msg.i> at an instant t, then u executes the following receiving action at t.

```
rcv <msg.i> ->
   <update local variables of u>;
   <may execute timeout-after <expression>>
```

Executing the receiving action of sensor u causes u to update its own local variables. It may also cause u to execute the statement "timeout-after <expression>". Note that executing the receiving action of sensor u does not cause u to send any message.

Let us summarize how the execution of a sensor network proceeds during one time instant t. First, the value of timer.u for every sensor u in the network is decremented by one at t. Second, if the value of any timer.u becomes 0 at t, then sensor u executes its timeout action at t. Execution of the timeout action of a sensor u at t assigns a new value to timer.u and may cause u to send one message at t. Third, if a sensor u sends a message at t, then any out-neighbor v of u may receive the message at t. Even if an out-neighbor v of u has executed its timeout action but sent no message at t, v can still receive u's message at t. In other words, a sensor may execute its timeout action followed by a receiving action at the same time instant provided that the sensor does not send a message during its execution of the timeout action. It follows from the above discussion that at a time instant, a sensor u executes exactly one of the following:

i. u sends one message, but receives no message.
ii. u receives one message, but sends no message.
iii. u sends no message and receives no message.

A *state* of a protocol is defined by a value for each variable, including the implicit variable timer.u, for each sensor u in the protocol. In every state, a value of each variable is selected from the declared domain of the variable. In our model, a state of a protocol can be changed by the execution of any (timeout or receiving) action in any sensor in the protocol.

5. Analysis methods

The model of sensor protocols presented in the previous section is rather detailed. Thus, we propose three analysis methods for analyzing a sensor protocol specification based on this model. In the first analysis, the correctness of the protocol specification is verified under two assumptions: idealized message transmission and no message collision. In the second analysis, the protocol specification is analyzed under the relaxation of the first assumption. In the third analysis, the protocol specification is analyzed under the relaxation of the first and second assumptions.

We refer to the first analysis as *nondeterministic analysis*, to the second analysis as *probabilistic analysis*, and to the third analysis as *collision analysis*.

In the next two sections, we present an example of a sensor protocol specification, and then analyze this protocol specification using our three analysis methods.

The two assumptions, of idealized message transmission and no message collision, upon which our analysis methods are based are stated as follows.

i. *Idealized message transmission:* In the topology of a sensor network, the probability label of each strong edge is 1 (instead of 0.95), and the probability label of each weak edge is 0 (instead of 0.5).
ii. *No message collision:* For every two distinct sensors u and v in a sensor network, if u is a (in- or out-) neighbor of v, or if the network has a third sensor w that is an out-neighbor for both u and v, then timer.u and timer.v have distinct values at every instant during the execution of the sensor network.

Some explanations concerning these two assumptions are in order. The first assumption has the effect of removing all the weak edges from the topology of a sensor network. It also has the effect of strengthening all the strong edges in a network topology.

To explain the second assumption, recall that a sensor u can send a message only during an execution of its timeout action, and that the timeout action of sensor u can be executed at an instant t iff the value of timer.u is 1 in the time unit $(t-1, t)$. Thus, the assumption of no message collision ensures that any two sensors, whose messages would collide if they were sent at the same instant, are guaranteed never to send messages at the same instant during any execution of the sensor network.

In order to make the second assumption, of no message collision, more acceptable, it is recommended that each statement "timeout-after x" in a sensor u be written as "timeout-after random(x, y)" where $x > 0$ and $x \leq y$. Thus, any new value assigned to timer.u is chosen uniformly from the range $x \cdots y$. If the new values of the timer variables are chosen uniformly from a reasonably large range, it is unlikely that any two timer variables will ever have the same value.

Next, we describe in some detail the three analysis methods.

i. Nondeterministic analysis:
This analysis is used to verify that a protocol is guaranteed to reach, from a given initial state, a desirable target state under the two assumptions of idealized message transmission and no message collision. For this analysis, we generate a state transition diagram of the protocol. In the diagram, each protocol state has one or more outgoing edges, since the protocol is specified using randomization steps of the form "timeout-after random(x, y)". From this diagram, we can verify that the protocol nondeterministically satisfies the desired reachability property.

ii. Probabilistic analysis:
This analysis is used to verify that a protocol will reach, from a given initial state, a desirable target state with a high probability, under the assumption of no message collision. For this analysis, we generate a probabilistic state transition diagram of the protocol, where each edge in the diagram is labeled with a probability. Note that the probabilities that label the edges in the probabilistic state transition diagram are computed from the probability labels in the network topology of the protocol. From this diagram, we can verify that the protocol probabilistically satisfies the desired reachability property.

iii. Collision analysis:
This analysis is used to analyze the effect of message collision and probabilistic message delivery on the performance and correctness of a sensor protocol. For this analysis, we simulate the execution of the protocol, and allow message collisions to occur. The nondeterministic and probabilistic analyses of a protocol can be carried out without specifying the values of x and y in the randomization steps "timeout-after random(x, y)" in the protocol specification. In choosing the values of x and y in these analyses, one needs to observe two restrictions. First, the difference $y-x$ should be large enough to ensure that the probability of message collision is reasonably small (and so the nondeterministic and probabilistic analyses of the protocol are reasonably accurate). Second, the difference $y-x$ should be small enough to ensure that the protocol reaches its desirable target state in a reasonably short time. To determine the appropriate values of x and y in the randomization steps, one can simulate the protocol for many value combinations of x and y and select the most appropriate values of x and y.

Each of the above three analyses is different from each other on its objective, complexity, and/or result quality. Performing the nondeterministic analysis on a protocol is simpler and easier than performing the probabilistic analysis on the protocol,

since the nondeterministic analysis assumes the idealized message transmission. However, the probabilistic analysis yields a more realistic result than that of the nondeterministic analysis. By applying the three analysis methods to a sensor protocol, sensor developers will be able to examine the influence of the unique characteristics of sensor networks on the performance of the protocol.

6. A protocol specification example

In this section, we use the model presented in Section 4 to specify a sensor protocol that can be used by any sensor in order to identify the strong neighbors of the sensor in the network. We refer to this protocol as the neighbor computation protocol. (Recall that two sensors in a network are strong neighbors iff there are two strong edges between them in the network topology.)

To identify the strong neighbors of a sensor u, u sends three request messages. Whenever a sensor v receives a request message sent by u, v sends a reply message. If sensor u receives two or more reply messages sent by the same sensor v, then u concludes that v is one of its strong neighbors.

Assume that the time period between two successive request messages sent by the same sensor is fixed. Under this assumption, if two neighboring sensors u and u' start to send their request messages at the same time, then the request messages sent by u will collide with the request messages sent by u' and both u and u' may end up concluding wrongly that they have no strong neighbors. Therefore, the time period between two successive request messages should be uniformly selected from a "large enough" range $1 \cdots x$. (In Section 7, we discuss how to choose a value for x.)

If every sensor v, that receives a request message from a sensor u, sends a reply message immediately after it receives the request message, then all the reply messages will collide with one another and u may end up receiving no reply messages. Thus, when a sensor v receives a request message from a sensor u, v should wait a random period of time before it sends a reply message. The length of this time period should be uniformly selected from the range $1 \cdots x$.

Consider the scenario where a sensor v receives a request message from a sensor u and decides to wait for some random period before it sends a reply message to u. It is possible that before v sends its reply to u, v receives another request message from another sensor u'. In this case, v should send one reply message to both u and u'. This requires that sensor v maintains a reply set, called *rset*, that contains the identifier of every sensor u from which v has received a request message and to which v has not yet sent a corresponding reply message. At the end of the above scenario, *rset* in sensor v has the value $\{u, u'\}$.

Note that sensors u and u' in the above scenario can be the same sensor u. Thus, *rset* in each sensor is a multiset rather than a set. For example, at the end of the above scenario, *rset* in sensor v has the value $\{u, u\}$.

Consider the scenario where a sensor u sends a request message and decides to wait for a random period before it sends its second request message. It is possible that before u sends its second request message, u receives a request message from another sensor u'. In this case, u should send one composite message that consists of the second request message and a reply message to sensor u'. We refer to this composite message as a request-reply message. In fact, every message in our protocol, whether a request message, a reply message, or a request-reply message, can be viewed as a request-reply message.

Each message in the neighbor computation protocol has three fields:

```
(v,b,s)
```

The first field v is the identifier of a sensor that sent this message. The second field b has two possible values: 0 and 1. If $b = 0$, then the message is a pure reply message. If $b = 1$, then the message is either a request message or a request-reply message. The third field s is the current value of *rset* in sensor v. Note that if the message is a pure request message, then $s =$ empty set.

Each sensor u has one constant x and eight variables, and has two actions as specified in Fig. 4. Variable *nghs* is the set of strong neighbors that sensor u needs to compute periodically. An element $rcvd[v]$ in variable $rcvd$ contains the number of replies that sensor u has received from sensor v after u has sent its first request message (in the current round of request messages). Variable *rm* stores the number of request messages that sensor u still needs to send (in the current round of request messages). Variable *rset* is the multiset of all the replies that sensor u needs to include in its next request-reply message. Variable *done* is a boolean variable whose value is true when and only when the current computation of the strong neighbors of sensor u is completed.

Initially, the value of *nghs* is the empty set, the value of every element in variable $rcvd$ is 0, the value of variable *rm* is 0, the value of variable *rset* is the empty set, the value of variable *done* is true, and the value of implicit variable timer.u is any value in the range $1 \cdots x$.

Sensor u executes its first action when the value of its timer.u becomes zero. The execution of this action starts by checking the value of *rm*. On one hand, if the value of *rm* is 0, then u recognizes that it does not need to send a request message, but it needs to send a reply message in case *rset* is non-empty. Thus, the sent message is of the form `(u,0,rset)`. Also if the value of *done* is true, then sensor u chooses arbitrarily whether it starts to compute its strong neighbors or not. If the value of *done* is false, sensor u invokes a procedure named `COMPNGH` that computes the strong neighbors of sensor u from array $rcvd$ and adds them to the set *nghs*. (In `COMPNGH`, a sensor v is computed to be a strong neighbor of u if $rcvd[v] \geq 2$.) On the other hand, if the value of *rm* is larger than 0, then u recognizes that it needs to send a request-reply message of the form `(u,1,rset)`.

```
sensor u      // sensor u where 0=< u < n
const x    : integer
var   nghs : set {u' | 0<= u' < n},       // strong ngh set
      rcvd : array [0 .. n-1] of 0..3,    // rcvd replies
      rset : set {u' | 0<= u' < n},       // reply set
      rm   : 0..3,                        // remaining request msgs
      done : boolean,                     // computation done or not
      v    : 0..n-1,                      // received sensor id
      b    : 0..1,                        // received request bit
      s    : set {u' | 0<= u' < n}        // received reply set
begin
   timeout-expires ->
       if rm=0 -> if rset != {} -> send (u,0,rset); rset := {}
                  [] rset = {}  -> skip
                  fi;
                  if done  -> skip               // no new round
                  [] done  -> nghs := {};        // start new round
                              rcvd := 0;
                              rm := 3;
                              done := false
                  [] !done -> COMPNGH(in rcvd, out nghs);
                              rcvd := 0;
                              done := true
                  fi; timeout-after random(1,x)
       [] rm>0 -> send (u,1,rset); rset := {};
                  rm := rm-1;
                  if rm>0 -> timeout-after random(1,x)
                  [] rm=0 -> timeout-after random(x+1,x+1)
                  fi
       fi
[] rcv (v,b,s) -> if !done -> rcvd[v] := rcvd[v] + NUM(u,s)
                  [] done  -> skip
                  fi;
                  if b=1 -> rset := rset+{v}
                  [] b=0 -> skip
                  fi
end
```

Fig. 4. Specification of sensor u in the neighbor computation protocol.

Sensor u executes the second action when u receives a `(v,b,s)` message sent by a neighboring sensor v. The execution of this action starts by checking the value of *done*. If the value of *done* is false, then the value of the element $rcvd[v]$ is incremented by $NUM(u, s)$, the number of times u occurs in the multiset s. Then sensor u checks the value of b in the received message. If the value of b is 1, then v is added to the multiset *rset*.

7. A protocol analysis example

In this section, we use the analysis methods outlined in Section 5 to analyze the correctness and performance of the neighbor computation protocol described in Section 6. Recall that there are three analysis methods: nondeterministic analysis, probabilistic analysis, and collision analysis. We apply each of the three analyses to the neighbor computation protocol in order.

7.1. Nondeterministic analysis

Nondeterministic analysis is used to show that the neighbor computation protocol satisfies some desirable progress property under the two assumptions of idealized message transmission and no message collision. The analysis is carried out from the point of view of a sensor u that needs to compute its strong neighbors.

From the assumption of idealized message transmission, each non-neighbor, weak neighbor or middle neighbor of u cannot receive any message sent by u, and/or cannot send any message to be received by u. Thus, non-neighbors, weak neighbors and middle neighbors of u have no effect on the computation carried out by u to identify its strong neighbors.

It remains to analyze the interaction between sensor u and each strong neighbor v of u. Fig. 5 shows the state transition diagram that describes the interaction between sensor u and its strong neighbor v. Each node in this diagram represents a state of the two sensors u and v. Each dashed edge represents the passing of real-time by one time unit. Each solid edge labeled u represents the execution of the timeout action in sensor u and the execution of the corresponding receiving action,

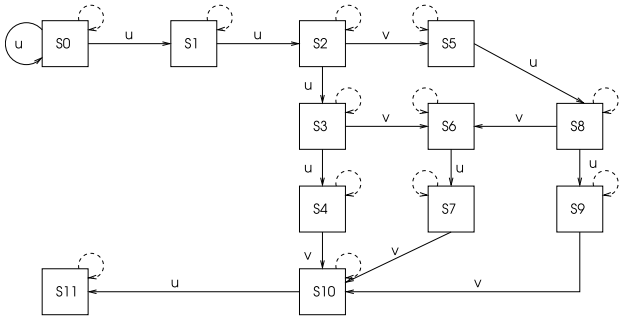

Fig. 5. State transition diagram.

S0:		rcvd(v).u=0 ∧	rm.u=0	∧ done.u=T ∧	NUM(u, rset.v)=0
S1:	nghs.u=∅ ∧	rcvd(v).u=0 ∧	rm.u=3	∧ done.u=F ∧	NUM(u, rset.v)=0
S2:	nghs.u=∅ ∧	rcvd(v).u=0 ∧	rm.u=2	∧ done.u=F ∧	NUM(u, rset.v)=1
S3:	nghs.u=∅ ∧	rcvd(v).u=0 ∧	rm.u=1	∧ done.u=F ∧	NUM(u, rset.v)=2
S4:	nghs.u=∅ ∧	rcvd(v).u=0 ∧	rm.u=0	∧ done.u=F ∧	NUM(u, rset.v)=3
S5:	nghs.u=∅ ∧	rcvd(v).u=1 ∧	rm.u=2	∧ done.u=F ∧	NUM(u, rset.v)=0
S6:	nghs.u=∅ ∧	rcvd(v).u=2 ∧	rm.u=1	∧ done.u=F ∧	NUM(u, rset.v)=0
S7:	nghs.u=∅ ∧	rcvd(v).u=2 ∧	rm.u=0	∧ done.u=F ∧	NUM(u, rset.v)=1
S8:	nghs.u=∅ ∧	rcvd(v).u=1 ∧	rm.u=1	∧ done.u=F ∧	NUM(u, rset.v)=1
S9:	nghs.u=∅ ∧	rcvd(v).u=1 ∧	rm.u=0	∧ done.u=F ∧	NUM(u, rset.v)=2
S10:	nghs.u=∅ ∧	rcvd(v).u=3 ∧	rm.u=0	∧ done.u=F ∧	NUM(u, rset.v)=0
S11:	nghs.u∋v ∧	rcvd(v).u=0 ∧	rm.u=0	∧ done.u=T ∧	NUM(u, rset.v)=0

Fig. 6. Specifying the states in the state transition diagram in Fig. 5.

if any, in sensor v. Each solid edge labeled v represents the execution of the timeout action in sensor v and the execution of the corresponding receiving action, if any, in sensor u.

Each of the states S0 through S11 in the state transition diagram is specified by a predicate in Fig. 6. Note that $\langle \text{var} \rangle.u$ is the value of variable $\langle \text{var} \rangle$ in sensor u, $rcvd[v].u$ is the value of element $rvcd[v]$ in array $rcvd$ in sensor u, and $NUM(u, rset.v)$ returns the number of times u occurs in the multiset $rset$ in sensor v.

From the state transition diagram, we conclude that the interaction between sensors u and v satisfies the following progress property.

State S1 eventually leads to state S11.

Therefore, the protocol is correct under the two assumptions of idealized message transmission and no message collision.

7.2. Probabilistic analysis

Probabilistic analysis is used to show that the neighbor computing protocol satisfies some desirable progress property, with a high probability, under the relaxation of the first assumption of idealized message transmission.

Under the assumption of idealized message transmission, the middle neighbors and weak neighbors of a sensor u play no role in u's computation of its strong neighbors. When this assumption is relaxed, this is no longer true, and a more refined analysis is in order.

Let u and v be distinct sensors in a network. If there are no edges or if there is exactly one edge between u and v in the network topology, then v has no effect on u's computation of its strong neighbors. Otherwise, let there be two edges between u and v in the network topology. Moreover, let p be the probability label of edge (u, v) and q be the probability label of edge (v, u), where $p, q > 0$. In this case, the probability that u identifies v as one of its strong neighbors is the probability that u receives at least two reply messages sent by v, which depends on the probability labels of edges (u, v) and (v, u), p and q.

Fig. 7 shows a part of the probabilistic state transition diagram that describes the interaction between sensor u and sensor v, where sensor u receives two or more reply messages sent by sensor v. Each solid edge labeled $u : prob$ represents the execution of the timeout action in sensor u and the execution of the corresponding receiving action, if any, in sensor v, and this transition occurs with probability $prob$. Similarly each solid edge labeled $v : prob$ represents the execution of the timeout action in sensor v and the execution of the corresponding receiving action, if any, in sensor u, and this transition occurs with probability $prob$. Each dotted edge represents a transition yielding to a state, where sensor u receives less than two reply messages sent by sensor v.

Each of the state S12 through S17 in the probabilistic state transition diagram is specified by a predicate in Fig. 8. The states of S0 through S11 are the same as in Fig. 6.

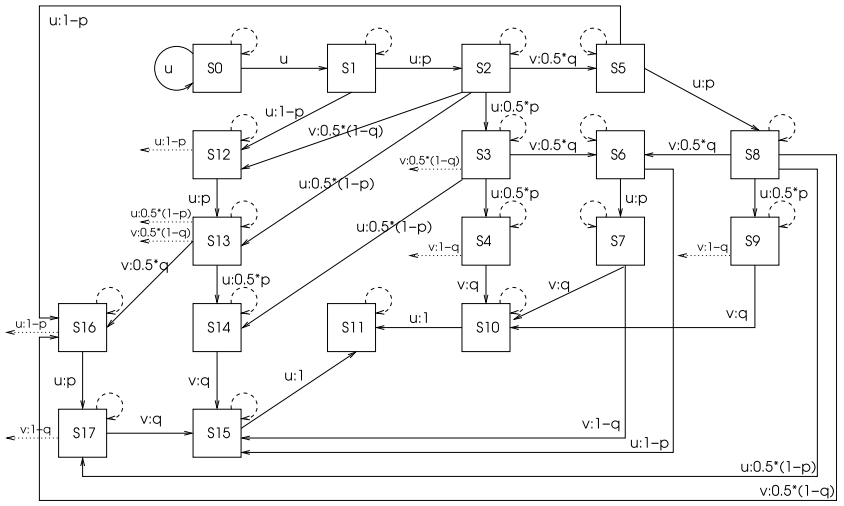

Fig. 7. Probabilistic state transition diagram.

S12:	nghs.u=$\emptyset$	$\wedge$	rcvd(v).u=0	$\wedge$	rm.u=2	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=0
S13:	nghs.u=$\emptyset$	$\wedge$	rcvd(v).u=0	$\wedge$	rm.u=1	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=1
S14:	nghs.u=$\emptyset$	$\wedge$	rcvd(v).u=0	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=2
S15:	nghs.u=$\emptyset$	$\wedge$	rcvd(v).u=2	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=0
S16:	nghs.u=$\emptyset$	$\wedge$	rcvd(v).u=1	$\wedge$	rm.u=1	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=0
S17:	nghs.u=$\emptyset$	$\wedge$	rcvd(v).u=1	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=1

Fig. 8. Specifying the states in the state transition diagram in Fig. 7.

Table 1
Probability that sensor u identifies sensor v as a strong neighbor in probabilistic analysis.

v	Strong neighbor	Middle neighbor	Weak neighbor	Non-neighbor
	0.949	0.472	0.195	0

In the probabilistic state transition diagram, sensor u identifies v as one of its strong neighbor if state S1 eventually leads to state S11. Thus, we need to compute the probability that state S1 eventually leads to state S11 through all different paths from S1 to S11, and there are 18 different paths from S1 to S11.

The probability to reach S11 from S1 through a path $(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_k)$, where s_i is a state in the diagram, $s_1 = S1$ and $s_k = S11$, is computed as $\prod_{i=1}^{k-1} e_{i,i+1}$ where $e_{i,i+1}$ is the probability to transit from s_i to s_{i+1}. For example, the probability to reach S11 from S1 through the path (S1, S12, S13, S14, S15, S11) is $(1-p) * p * (0.5 * p) * q$.

Recall that if v is a strong neighbor of u, then both p and q equal 0.95, if v is a middle neighbor of u, then one of the two probabilities is 0.95 and the other is 0.5, and if v is a weak neighbor of u, then both p and q equal 0.5. Substituting these values of p and q in the diagram, we obtain the results in Table 1. (Note that the result for a middle neighbor is the average of the two cases of $p, q = 0.95, 0.5$ and $p, q = 0.5, 0.95$.)

In the above diagram, some states such as S2, S3, S8, and S13 lead to next states either by the execution of the timeout action in u or by the execution of the timeout action in v. Note that the above analysis is based on the assumption that in such states, the protocol transits to a next state by the execution of the timeout action in u with probability 0.5 and by the execution of the timeout action in v with probability 0.5. However, this assumption is not accurate. The transitions to states S2, S3, S8, and S13, are caused by the execution of the timeout action in u, and so the probability of transition from each of these states to a next state by the execution of the timeout action in v may be higher than that by the execution of the timeout action in u. Thus, in the above analysis, the probabilities to reach S11 through some paths (for example, path (S1, S2, S5, S8, S6, S7, S10, S11)) are computed slightly lower than those that can be obtained from observing an execution of the protocol. However, the probabilities to reach S11 through other paths (for example, path (S1, S2, S3, S4, S10, S11)) are computed slightly higher than those that can be obtained from observing an execution of the protocol.

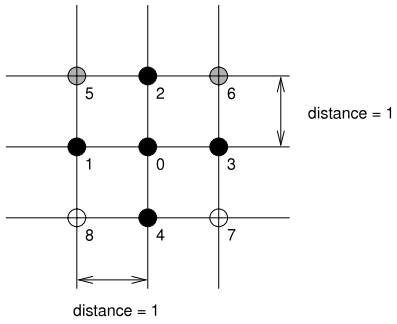

Fig. 9. The simulated network.

Table 2
Probability that sensor 0 identifies sensor v as a strong neighbor in collision analysis.

v	1	2	3	4	Avg (1–4)	5	6	Avg (5–6)	7	8	Avg (7–8)
$x = 64$	0.81	0.835	0.851	0.826	0.831	0.413	0.376	0.395	0.132	0.153	0.143
$x = 128$	0.888	0.877	0.897	0.898	0.89	0.435	0.404	0.42	0.157	0.157	0.157
$x = 256$	0.918	0.94	0.935	0.94	0.933	0.468	0.454	0.461	0.169	0.19	0.18

7.3. Collision analysis

The above probabilistic analysis is based on the assumption of no message collision. In order to take into account the effects of message collision on the execution of a sensor network, one better simulates the network execution and allows message collisions to occur in the simulation.

We developed a simulator that can simulate the execution of the neighbor computation protocol based on our formal model described in Sections 3 and 4. Note that the simulator used in this section monitors the execution of an abstract and generic version of a sensor protocol (rather than any specific implementation of the protocol). However, our model actually captures unique characteristics of sensor networks such as local broadcast, probabilistic message transmission, and message collision. Thus, this simulator is useful to evaluate the performance of sensor protocols, and explore the implications of design decisions on the protocols.

In this simulator, the simulated network consists of nine sensors, sensor 0 through sensor 8, as shown as in Fig. 9. We assume that only sensor 0 in this network needs to periodically identify its strong neighbors. We also assume that sensor 0 has four strong neighbors, namely sensors 1 through 4, two middle neighbors, namely sensors 5 and 6, and two weak neighbors, namely sensors 7 and 8.

The "neighboring relation" over the sensors 1 through 8 is determined by the following three rules.

i. If the distance between two sensors is at most 1, the edge is labeled with probability 0.95 (i.e. a strong edge).
ii. If the distance between two sensors is larger than 1 and less than 2, the edge is labeled with probability 0.5 (i.e. a weak edge).
iii. If the distance between two sensors is at least 2, there is no edge between the two sensors.

In order to run the simulation of the neighbor computation protocol, we need to choose the value x. There are two contradictory concerns that can affect our choice of x. If x is large, say 1024, the probability of message collision becomes small, and consequently the probability of correctly identifying a strong neighbor, as measured from the simulation, becomes close to the same probability, as estimated from the probabilistic analysis. On the other hand, if x is large, the average execution time of the protocol, which is around $2 * x + 1$ time units, becomes large.

To choose an appropriate value for x, we ran the simulation three times using different values of x: 64, 128, and 256. In each simulation run, we measured the probability of correctly identifying a strong neighbor. Each of the simulation results is the average of 1000 runs. In this work, we chose the smallest value of x whose simulation run yielded the probability of correctly identifying a strong neighbor to become around 0.9. From Table 2, we conclude that such value for x is 128.

From the simulation run where $x = 128$, we get the following results. For a strong neighbor v, sensor 0 identifies v as a strong neighbor with probability 0.89. For a middle neighbor v, sensor 0 identifies v as a strong neighbor with probability 0.42. For a weak neighbor v, sensor 0 identifies v as a strong neighbor with probability 0.157. (Note that by choosing $x = 128$, the average execution time of the protocol is around 26 s, under the assumption that each time unit is 100 ms.)

Using simulation, the influence of probabilistic message delivery and message collision on the correctness and performance of the neighbor computation protocol was analyzed, and the most appropriate value for x was selected. Performance evaluation based on a specific implementation of the protocol, such as TinyOS [26], is beyond the scope of this paper.

Table 3
Probability that sensor u identifies sensor v as a strong neighbor.

Type	Strong ngh	Middle ngh	Weak ngh	Non-ngh
Nondeterministic analysis	1	0	0	0
Probabilistic analysis	0.949	0.472	0.195	0
Collision analysis	0.89	0.42	0.157	0

As a summary, Table 3 shows the probability that a sensor u identifies another sensor v as a strong neighbor in each of the analysis methods. As we relax the two assumptions of idealized message transmission and no message collision one by one, the probability that u correctly identifies a strong neighbor v is decreased. Moreover, middle neighbors and weak neighbors of u affect u's computation of its strong neighbors.

8. A property verification example

The state-based model presented in this paper can be used to verify that a sensor protocol satisfies some desirable properties, such as self-stabilization, time-related, safety, progress, and security properties. In this section, we show that the neighbor computation protocol described in Section 6 is self-stabilizing.

A protocol is called *self-stabilizing* if starting from any state, the protocol eventually converges to a *legitimate* state where it can perform its intended function correctly, and once it reaches a legitimate state, the execution of the protocol yields in a legitimate state [39,40]. Note that self-stabilization is desirable for protocols in a sensor network that suffers from various faults such as memory corruption, wrong initialization, and hardware/software failure.

Recall that a state of a protocol is defined by a value for each variable, including the implicit variable $timer.u$, for each sensor u in the protocol. In every state (whether legitimate or illegitimate) a value of each variable is selected from the declared domain of the variable. The values of variables in sensor u, such as $nghs$, $rcvd$, $rset$, rm, $done$, and $timer.u$, can be corrupted arbitrarily by some fault, resulting that the protocol reaches an illegitimate state.

The neighbor computation protocol is self-stabilizing such that starting from any illegitimate state, the protocol eventually converges to a legitimate state where every sensor can correctly compute its strong neighbors in the network, and continues to execute within legitimate states. We prove this property of the protocol from the point of view of a sensor u that needs to compute its strong neighbors. Note that any sensor in the network can be one that needs to compute its strong neighbors.

A state S of the protocol is *legitimate* iff either S is a state where the predicate P

$$timer.u = 1 \wedge rm.u = 0 \wedge done.u \wedge$$
$$(\text{for every neighbor } v \text{ of } u, \text{NUM}(u, rset.v) = 0)$$

holds or S is a state that is reachable from a state, where this predicate P holds, by some execution of the protocol.

It follows from the above definition that if the protocol starts from a legitimate state, then every time sensor u starts a new round, sensor u resets $nghs.u$ to an empty set, $rcvd.u$ to 0, and $rm.u$ to 3, and each (in- or out-) neighbor v of u does not have any reply message to u for a previous round. Therefore, in the new round, sensor u can compute correctly whether each v is a strong neighbor of u or not.

For the proof, we assume that message delivery is probabilistic and message collision can occur in the network, and so request messages sent by u and reply messages sent by each v can be lost. Also we assume that in every state (whether legitimate or illegitimate), the value of $timer.u$ for every sensor u is at most $x + 1$ time units. Note that this is maintained by the execution of the neighbor computation protocol.

Lemma 1. *Starting from any illegitimate state, every sensor u, which has a corrupted value of rset.u, sends a reply message with the corrupted value, and resets rset.u to an empty set within* $x + 1$ *time units.*

Proof. The protocol can start from a state in which some sensor u has a corrupted value of $rset.u$. Sensor u times-out within $x + 1$ time units, since the value of $timer.u$ is at most $x + 1$ time units by our assumption. When u times-out, it sends a reply message based on a corrupted value of $rset.u$, and resets $rset.u$ to an empty set. Thus, after every sensor u resets $rset.u$ within $x + 1$ time units, u has a legitimate value of $rset.u$, and so does not send any reply message with a corrupted value of $rset.u$. □

Lemma 2. *Starting from any illegitimate state, every illegitimate neighbor computation of a sensor is terminated within* $5 * x + 2$ *time units.*

Proof. Consider a case that some neighbor v of a sensor u times-out and sends a reply message with a corrupted value of $rset.v$ with respect to u at time t. It is possible that u times-out at time t', before v times-out at t, where $t' < t$, and computes its strong neighbors illegitimately, if $rm.u = 0$ and $done.u = false$ initially. In this case, at time t', u finishes one round of the illegitimate neighbor computation, but v still has a corrupted value of $rset.v$ for u.

Assume that at time t, sensor u times-out again and starts a new round, resetting $nghs.u$, $rcvd.u$, $rm.u$, and $done.u$ to an empty set, 0, 3, and false, respectively. Also assume that u selects x, which is the maximum possible value by legitimate

execution at t, for $timer.u$. Since $done.u$ becomes false at time t, if u receives the reply message sent by v at t, u increases the number of reply messages sent by v based on the corrupted value of $rset.v$. At time $t + x$, u sends the first request message, and can select x for $timer.u$. At time $t + 2 * x$, u sends the second message, and can select x for $timer.u$. At $t + 3 * x$, u sends the third message, and assigns $rm.u$ 0. Since $rm.u = 0$, u assigns $timer.u$ $x + 1$. At time $t + 4 * x + 1$, u finally computes its strong neighbors. The computed strong neighbors of u at time $t + 4 * x + 1$ may not be correct, since u may receive the corrupted $rset.v$ for u at time t. When u finishes another round of the illegitimate neighbor computation at $t + 4 * x + 1$, v does not have a corrupted value of $rset.v$ (since v resets the corrupted value at t by Lemma 1), and so the number of reply messages in v to u is 0. Thus, u will not have a new round of the illegitimate neighbor computation based on a corrupted value of $rset.v$. Therefore, the last illegitimate neighbor computation of u is terminated within $5 * x + 2$ time units, since $t \leq x + 1$ by Lemma 1. □

Theorem 1. *Starting from any illegitimate state, the protocol reaches a legitimate state within* $6 * x + 2$ *time units, and continues to execute within legitimate states.*

Proof. Consider a case that the last illegitimate neighbor computation of some sensor u terminates at time t. Once the last illegitimate computation is done, the values of $rm.u$ and $done.u$ are 0 and true, respectively, and for every neighbor v of u, the number of reply messages in v with respect to u is 0 (as shown in the proof of Lemma 2). Assume that u selects x (which is the maximum possible value by legitimate execution) for $timer.u$ at t. In time unit $(t + x - 1, t + x)$, $timer.u$ becomes 1, and predicate P holds. When u times-out at time $t + x$, u can start a new round, and compute its strong neighbors correctly. Therefore, starting from any state, the neighbor computation protocol converges to a legitimate state within $6 * x + 2$ time units, since $t \leq 5 * x + 2$ by Lemma 2. □

9. Discussion

Analysis process. The analysis process of a sensor protocol is still hard. In Section 7, the nondeterministic and probabilistic state transition diagrams of the neighbor computation protocol were generated by our thorough inspection of the protocol behavior. These state transition diagrams are focused on the interaction between two sensors u and v from the view point of sensor u that needs to compute its strong neighbors. Thus, the state transition diagrams only contain a small number of variables that have effects on this interaction, and result in having a small number of states. However, many sensor protocols, especially for large scale networks, will require to have a large number of states. In these cases, it may be infeasible to generate a state transition diagram from a protocol specification by inspection.

In this paper, we focused on investigating a model and analysis methods that are suitable for sensor protocols. We consider the study on how to automate or facilitate the analysis methods as a subject of our future work. Next, we discuss possible approaches and issues on this subject.

An algorithm or tool that generates a (nondeterministic and probabilistic) state transition diagram automatically from a protocol specification will facilitate the nondeterministic and probabilistic analyses of the protocol. Thus, we can use an existing verification tool or model checker that has been used to analyze network protocols. For the nondeterministic and probabilistic analyses, a verification tool should be able to model and analyze systems that exhibit nondeterministic and probabilistic behavior. (One example of such verification tools is PRISM [41].) A verification tool can automatically verify that a described system eventually reaches a certain state, or analyze the probability of the system reaching a certain state, based on exhaustive search [41].

To use a verification tool for the analyses, the following issues need to be considered and solved.

- *Reducing the state space of a sensor protocol*: Due to the state space explosion, analyzing a protocol for a large scale network can exceed the ability of a verification tool. Thus, in the literature, a small number of nodes (less than 10 nodes) have been considered for verification [8,21,22,30].

  One approach to reduce the state space is to focus on key interaction of a protocol. In [22], a few scenarios among four or six nodes, which could cause problems, were selected to verify a routing protocol in ad-hoc networks. In [21], the functionality of the AODV protocol was refined into the sender, intermediate node and destination, while preserving the key behavior of the protocol. For verification, one sender, some number of intermediate nodes, and one destination were only considered, which is similar to our approach focusing on a sensor u that computes its strong neighbors and a neighbor v of u.

  In the above approach, the generic specification of a protocol might need to be refined into several specifications. Each specification is for a specific function such as the sender in the AODV protocol. Note that the manual refinement of a protocol specification is not very difficult for a sensor protocol developer who writes the original specification.
- *Translating a protocol from our specification to a modeling language of a verification tool*: A protocol specification (or refined specifications) needs to be translated into the corresponding code in a modeling language of a verification tool. An automatic translation tool will expedite the process. On developing a translation tool, we need to adapt the verification tool properly such that it can model the characteristics of sensor networks such as local broadcast and asymmetric communication as in [22,30]. For the nondeterministic analysis, the translation tool needs to generate code that models the nondeterministic behavior of a sensor protocol under the assumptions of idealized message transmission and no message collision. On the other hand, for the probabilistic analysis, it needs to generate code that models the probabilistic behavior under the assumption of no message collision.

For the collision analysis, a simulation code generator which generates executable simulation code automatically from a protocol specification will facilitate the analysis process. The collision effect on a sensor protocol is different depending on a given network topology as well as the values of parameters for randomization steps. This generator will allow us to analyze the protocol for various network topologies with different values of the parameters. Moreover, it can be used to analyze the protocol for a large scale sensor network, composed of hundreds or thousands of sensors, unlike a verification tool.

In this work, the simulation code was generated using C, but other programming languages such as C++, Java, and MATLAB can be used. A generated simulation code will include the following modules for a given protocol and topology: an initialization module that initializes variables in sensors, a sending module that sends messages among sensors according to the given topology, a main loop module in which for a time instant, enabled timeout actions and then enabled receiving actions of sensors are executed in order, and finally the time passes to the next time instant.

Note that probabilistic message transmission can be implemented by comparing a generated random number with a threshold, computed based on the probability label of an edge in the topology. Message collision can be implemented by enabling a receiving action of a sensor only when its total number of received messages at a time instant is one.

Complex sensor protocols. The example of the neighbor computation protocol given in this paper is rather simple, since the protocol describes the execution of sensors within one-hop communication. However, our model can be used for various, more complex, sensor protocols. For instance, a version of a flood protocol is formally specified based on the model in [42], and a routing protocol that builds a routing tree using a logical grid is formally specified based on a similar model in [43]. Also the self-stabilization properties of these protocols are verified. In these protocols, two specifications were given, one for the base station that initiates floods or for the root of a tree, and the other for all other sensors, while only one specification was given to describe the neighbor computation protocol. By describing the behavior of each sensor in a multi-hop network, the complex interactions between the sensors can be specified, and their properties can be verified based on states of the protocol, which are defined by a value for each variable for each sensor in the protocol.

10. Concluding remarks

In this paper, we presented a state-based model of sensor network protocols. This model accommodates several characteristics of sensor networks, such as unavoidable local broadcast, probabilistic message transmission, asymmetric communication, message collision, and timeout actions and randomization steps. We also proposed three analysis methods of sensor protocols, nondeterministic analysis, probabilistic analysis, and collision analysis. These methods can be used to analyze a sensor protocol specified in the state-based model under different assumptions about message transmission and collision. Using the model, we can also prove that a sensor protocol satisfies some desirable properties, such as self-stabilization, time-related, safety, progress, and security properties.

Although the probability label of a strong edge is chosen to be 0.95 and the probability label of a weak edge is chosen to be 0.5 in this work, different values can be chosen for these probability labels for different settings of sensor networks.

There are several directions to extend our model for sensor protocols. First, our model assumes that sensors in a sensor network are stationary. The model can be extended to support a sensor network with mobile sensors. Second, energy models of sensors can be added to our model to estimate the lifetime of a sensor network or measure the amount of energy consumed by a sensor.

References

[1] M. Gouda, Y. Choi, A state-based model of sensor protocols, in: Proceedings of 9th International Conference on Principles of Distributed Systems, OPODIS 2005, 2005.
[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, M. Miyashita, A line in the sand: a wireless sensor network for target detection, classification, and tracking, Computer Networks (Elsevier) 46 (5) (2004) 605–634.
[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Computer Networks Elsevier Science 38 (4) (2002) 393–422.
[4] A. Mainwaring, J. Polastre, D. Culler, J. Anderson, Wireless sensor networks for habitat monitoring, in: Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, GA, 2002.
[5] A. Hall, Seven myths of formal methods, IEEE Software 7 (1990) 11–19.
[6] D. Câmara, A.A. Loureiro, F. Filali, Methodology for formal verification of routing protocols for ad hoc wireless networks, in: GLOBECOM'07: Proceedings of the 50th IEEE Global Telecommunications Conference, 2007, pp. 705–709.
[7] D. Câmara, A.A. Loureiro, F. Filali, Formal verification of routing protocols for wireless ad hoc networks, in: Guide to Wireless Ad Hoc Networks, Computer Communications and Networks, Springer, London, 2009, pp. 1–22.
[8] C. Xiong, T. Murata, J. Tsai, Modeling and simulation of routing protocol for mobile ad hoc networks using colored petri nets, in: CRPIT'02: Proceedings of the Conference on Application and Theory of Petri Nets, Australian Computer Society, Inc, Darlinghurst, Australia, 2002, pp. 145–153.
[9] M. Demirbas, A. Arora, M. Gouda, A Pursuer-Evader game for sensor networks, in: Sixth Symposium on Self-Stabilizing Systems, SSS'03.
[10] M. Demirbas, A. Arora, T. Nolte, N. Lynch, A hierarchy-based fault-local stabilizing algorithm for tracking in sensor networks, in: 8th International Conference on Principles of Distributed Systems, OPODIS, 2004.
[11] D. Bein, A. Datta, A self-stabilizing directed diffusion protocol for sensor networks, in: Proceedings of the 2004 International Conference on Parallel Processing Workshops, ICPPW'04, 2004.
[12] S. Bapat, A. Arora, Stabilizing reconfiguration in wireless sensor networks, in: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing, SUTC, 2006.
[13] S. Kulkarni, M. Arumugam, Transformations for write - all - with - collision model, in: Dependable Wireless Sensor Networks, Computer Communications (Elsevier) 29 (2) (2005) 183–199. special issue.

[14] T. Herman, Models of self-stabilization and sensor networks, in: Proceedings of IWDC 2003, in: LNCS, vol. 2981, Springer Verlag, 2003.
[15] T. Herman, S. Tixeuil, A distributed tdma slot assignment algorithm for wireless sensor networks, in: Proceedings of the First Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'2004), no. 3121, in: Lecture Notes in Computer Science, Springer-Verlag, Turku, Finland, 2004, pp. 45–58.
[16] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, A. Wood, EnvioTrack: towards an environmental computing paradigm for distributed sensor networks, in: Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan, 2004.
[17] R. Newton, M. Welsh, Region streams: functional macroprogramming for sensor networks, in: International Workshop on Data Management for Sensor Networks, DMSN, VLDB 2004, 2004.
[18] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TAG: a Tiny AGgregation service for ad-hoc sensor networks, ACM SIGOPS Operating Systems Review 36 (Winter) (2002) 131–146.
[19] J. Liu, M. Chu, J. Liu, J. Reich, F. Zhao, Distributed state representation for tracking problems in sensor networks, in: IPSN 2004, 2004.
[20] Y. Yao, J. Gehrke, The cougar approach to in-network query processing in sensor networks, ACM SIGMOD Record 31 (3).
[21] S. Chiyangwa, M. Kwiatkowska, A timing analysis of aodv, in: M. Steffen, G. Zavattaro (Eds.), Formal Methods for Open Object-Based Distributed Systems, in: Lecture Notes in Computer Science, vol. 3535, Springer, Berlin, Heidelberg, 2005, pp. 306–321.
[22] O. Wibling, J. Parrow, A. Pears, Automatized verification of ad hoc routing protocols, in: D. de Frutos-Escrig, M. Nunez (Eds.), Formal Techniques for Networked and Distributed Systems - ORTE 2004, in: Lecture Notes in Computer Science, vol. 3235, Springer, Berlin, Heidelberg, 2004, pp. 343–358.
[23] N. Kothari, T. Millstein, R. Govindan, Deriving state machines from tinyos programs using symbolic execution, in: Proceedings of the international conference on Information Processing in Sensor Networks, IPSN 08, 2008, pp. 271–282.
[24] R. Alur, P. Černý, P. Madhusudan, W. Nam, Synthesis of interface specifications for java classes, in: POPL'05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2005, pp. 98–109.
[25] G. Ammons, R. Bodík, J.R. Larus, Mining specifications, SIGPLAN Notices 37 (1) (2002) 4–16.
[26] Tinyos, http://www.tinyos.net.
[27] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, R. Han, Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms, Mobile Network Applications 10 (2005) 563–579.
[28] M. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, S. Olivieri, A framework for modeling, simulation and automatic code generation of sensor network application, in: Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON'08, 2008, pp. 515–522.
[29] L. Vieira, B. Vitorino, M. Vieira, D. Silva, A. Fernandes, Wisdom: a visual development framework for multi-platform wireless sensor networks, in: Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation, ETFA'05, 2005.
[30] A. Fehnker, P. Gao, Formal verification and simulation for performance analysis for probabilistic broadcast protocols, in: Proc. 5th International Conference on Ad-Hoc, Mobile, and Wireless Networks, ADHOC-NOW'06, in: LNCS, vol. 4104, Springer, 2006, pp. 128–141.
[31] Y. Choi, M.G. Gouda, M.C. Kim, A. Arora, The mote connectivity protocol, in: Proceedings of 12th International Conference on Computer Communications and Networks, ICCCN 2003, Dallas, TX, 2003, pp. 533–538.
[32] Crossbow Technology Inc., http://www.xbow.com/.
[33] A. Woo, T. Tony, D. Culler, Taming the underlying challenges of reliable multihop routing in sensor networks, in: Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems, Los Angeles, CA, 2003.
[34] A. Cerpa, N. Busek, D. Estrin, SCALE: a tool for simple connectivity assessment in lossy environments, CENS Technical Report 21.
[35] J. Zhao, R. Govindan, Understanding packet delivery performance in dense wireless sensor networks, in: Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems, Los Angeles, CA, 2003.
[36] F. Kuhn, A. Zollinger, Ad-hoc networks beyond unit disk graphs, in: DIALM-POMC'03: Proceedings of the 2003 joint workshop on Foundations of mobile computing, ACM, 2003, pp. 69–78.
[37] F. Kuhn, T. Moscibroda, R. Wattenhofer, Initializing newly deployed ad hoc and sensor networks, in: MobiCom '04: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking, ACM, 2004, pp. 260–274.
[38] L. Barriére, P. Fraigniaud, L. Narayanan, Robust position-based routing in wireless ad hoc networks with unstable transmission ranges, in: DIALM'01: Proceedings of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, ACM, 2001, pp. 19–27.
[39] A. Arora, M. Gouda, Closure and convergence: a foundation of fault-tolerant computing, IEEE Transactions on Software Engineering 19 (11) (1993) 1015–1027.
[40] S. Dolev, Self-Stabilization, The MIT Press, 2000.
[41] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: G. Gopalakrishnan, S. Qadeer (Eds.), Proc. 23rd International Conference on Computer Aided Verification, CAV'11, in: LNCS, vol. 6806, Springer, 2011, pp. 585–591.
[42] Y. Choi, C.-T. Huang, M. Gouda, Stabilization of flood sequencing protocols in sensor networks, Parallel and Distributed Systems, IEEE Transactions on 21 (7) (2010) 1042–1055.
[43] Y. Choi, M. Gouda, H. Zhang, A. Arora, Stabilization of grid routing in sensor networks, AIAA Journal of Aerospace Computing, Information, and Communication 3 (2006) 214–233.