A State-based Model of Sensor Protocols

Mohamed G. Gouda and Young-ri Choi *

Department of Computer Sciences, The University of Texas at Austin,
1 University Station C0500, Austin, Texas 78712-0233, U.S.A.
{gouda, yrchoi}@cs.utexas.edu

Abstract. We introduce a state-based model that can be used in specifying sensor network protocols. This model accommodates several features that are common in sensor networks. Examples of these features are 1-step local broadcast, probabilistic delivery of messages, asymmetric communication, and message collision. We propose a three-step method for verifying sensor protocols that are specified in this model. In the first step, the specified 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 step, the protocol is proven "probabilistically correct" under the assumption that message delivery is probabilistic but message collision is guaranteed not to occur. In the third step, the correctness of the protocol is proven by a simulation where message delivery is probabilistic and message collision may occur (when two or more neighboring sensors happen to send messages at the same time). 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 verification method to the protocol.

Keywords Sensor network, State-based model, Formal model, Protocol specification, Protocol verification

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 [1], disaster monitoring [2] and habitat monitoring [3].

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.

* Young-ri Choi is the corresponding author of this paper.

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.
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 protocols need to heavily depend on timeout actions and randomization steps to perform their functions.

The above characteristics of 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 towards formal specification, verification, and design of these protocols.

There have been earlier efforts to model the software of sensor networks. Examples of these efforts are [4], [5], [6], and [7]. We review these and other efforts in the related work section of this paper. Nevertheless, it is important to state here that all these efforts are not directed towards modeling sensor protocols; rather they 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, asymmetric communication, and message collision, 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.

2 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 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 that label the edges in the topology of a sensor network. These two probabilities are 0.95 and 0.5 in this work. (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 [8].)

In the topology of a sensor network, an edge that is labeled with the large probability 0.95 is called a *strong edge*, and an edge that is labeled with the small probability 0.5 is called a *weak edge*.

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.

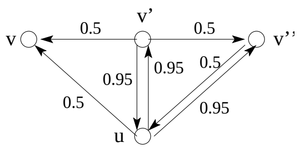

Fig. 1. Topology of a sensor network

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''.

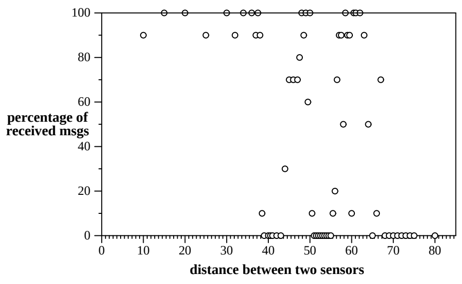

Fig. 2. Percentage of received messages

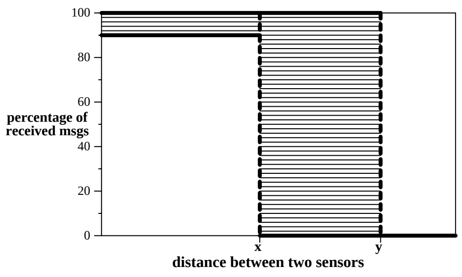

Fig. 3. Idealized percentage of received messages

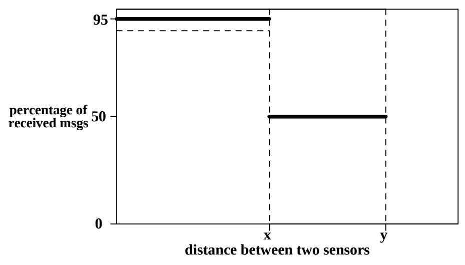

Fig. 4. The probability label of an edge

In [8], we describe some experiments that we have carried out using Mica sensors [9]. 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 reported in [10] and [11].)

We observe that from Fig. 2 if the distance between two sensors u and v is in the range 0 .. 38 inches, v receives between 90% and 100% of the messages sent by u. One the other hand, if the distance between sensors u and v is in the range 38 .. 67 inches, v receives anywhere between 0% and 100% of the messages sent by u. Finally, the distance between sensors u and v is longer than 67 inches, v receives 0% of the messages sent by u. From these observations, the diagram in Fig. 2 can be "idealized" as shown in Fig. 3.

Let u and v be distinct sensors in the topology of a sensor network, and assume there is a directed edge from u to v in the network topology. According to the idealized diagram in Fig. 3, if the distance between u and v is in the range 0 .. x, then v receives between 90% and 100% of the messages sent by u. Thus, the directed edge from u to v can be labeled with a probability 0.95, and the edge is strong. If the distance between u and v is in the range x .. y, then v receives between 0% and 100% of the messages sent by u. Thus, the directed edge from u to v can be labeled with a probability 0.5, and the edge is weak. Fig. 4 shows how the probability label of an edge from one sensor to another in the network topology is chosen based on the distance between the two sensors.

3 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 follows:

```
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
```

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 milliseconds.)

At a time instant t, if the timeout of a sensor u expires, then u executes its timeout action (at t). 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, the value of timer.u in the time unit $(t,\, t+1)$ is the value of <expression> in the same time unit.

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 t, provided that the following three conditions hold.

i. A random integer number is uniformly selected in the range 0 .. 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 .. 99, and this selected number is at least $100 * p'$, where p' is the probability label of edge (w,v) in the network topology.

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>>
```

Note that 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>" which causes the timeout of u to expire after k time units, where k is the value of <expression> in the time unit (t, $t+1$). 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.

In the remainder of the paper, we use this model to specify an example protocol and to prove the protocol correct utilizing our verification method described in the next section.

4 Three-step Verification Method

The model of sensor network protocols presented in the previous sections is rather complicated. Thus, the correctness of a sensor protocol specification, that is based on this model, is better verified in steps, in fact three steps. In the first step, the correctness of the protocol specification is verified under two assumptions: idealized message transmission and no message collision. In the second step, the effect of relaxing the first assumption on the established correctness in the first step is analyzed. In the third step, the effect of relaxing the second assumption on the established correctness in the second step is analyzed.

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

In the next two sections, we present an example of a sensor protocol specification, and then verify the correctness of this specification using our verification method.

The two assumptions, of idealized message transmission and no message collision, upon which our verification method is 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 .. y. Because 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 steps of our verification method.

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. Simulation:
The nondeterministic and probabilistic analyses (in the first two steps) 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 steps, 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.

5 A Protocol Specification Example

In this section, we use the above model to specify a sensor protocol that can be used by any sensor in order to identify the strong neighbors of that sensor in its 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, sensor u sends three request messages. Whenever a sensor v receives a request message sent by sensor u, sensor v sends a reply message. If sensor u receives two or more reply messages sent by the same sensor v, then sensor u concludes that sensor 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 .. x. (In the next section, 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 .. 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 sensor v 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 as follows.

```
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
```

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 .. x.

Each sensor u has two actions that are specified as follows.

```
sensor u      // sensor u where 0=< u < n

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
```

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 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*.

6 A Protocol Verification Example

In this section, we use the verification method outlined in Section 4 to verify the correctness of the neighbor computation protocol in Section 5. Recall that the verification method consists of three steps: nondeterministic analysis, probabilistic analysis, and simulation. We apply each of these steps to the neighbor computation protocol in order.

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, discussed above. 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, 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.

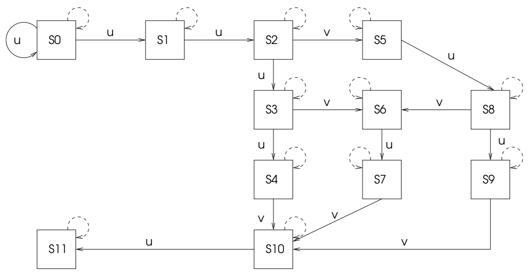

Fig. 5. State Transition Diagram

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

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

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, 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 $rcvd[v].u$ is the value of element $rvcd[v]$ in array $rcvd$ in sensor u, $rm.u$ is the value of variable rm in sensor u, $done.u$ is the value of variable $done$ 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.

Probabilistic analysis is used to analyze the effect of relaxing the first assumption of idealized message transmission on the correctness and performance of the protocol. 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. Let u and v are 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). In this case, the probability that u identifies v as one of its strong neighbors depends on the probability labels of edges (u,v) and (v,u), p and q.

Simulation is used to analyze the effect of relaxing the two assumptions of idealized message transmission and no message collision on the correctness and performance of the protocol. In order to run the simulation of the protocol, we need to choose the value of x. There are two contradictory concerns that can affect our choice of x. If x is large, 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. Thus, the simulation is used to evaluate the performance of the protocol over various values of x and choose the most appropriate value for x.

As we relax the two assumptions one by one, the probability for a sensor u to identify a strong neighbor v is decreased. Moreover, middle neighbors and weak neighbors of u affect u's computation of its strong neighbors. The details of probabilistic analysis and simulation can be found in [12].

7 Related Work

Several models for sensor applications have been proposed [5], [6], [4], [7]. In general, the purpose of these models is to hide application programmers from low-level details such as routing, group management, resource management, etc. EnvioTrack [5] provides a high-level programming abstraction for tracking applications in sensor networks. Newton and Welsh proposed a functional language to specify the global behavior of a sensor application [6]. Liu et al. presented a state-centric programming model for sensor networks [4]. Database approach was proposed in TAG [7]. Unlike these models, our proposed model is to describe sensor protocols (that are responsible for routing, group management, etc). Thus, our model deals with the intricate characteristics of wireless sensor networks described in Section 1.

Levis et al. developed a communication-centric virtual machine for sensor networks called Maté [13]. Using Maté's high-level interfaces, sensor applications can be composed in a very short code.

The Abstract Protocol notation was developed earlier to specify network protocols in traditional networks [14]. Gracanin et al. proposed a model that focuses on services provided by wireless sensor networks [15]. Volgyesi et al. proposed a model to describe interface specification of components for sensor networks [16]. This model allows us to check the compatibility of components and to verify the design and composition of components based on their interfaces. In [17], antireplay protocols for sensor networks were proposed. Also it was shown that the proposed protocols satisfy desirable properties (such as corruption detection, replay detection, and freshness detection) under some assumption. In this paper, we investigate what a model should be to describe sensor protocols and how sensor protocols specified in this model can be verified.

Several simulation frameworks have been developed for sensor networks [18], [19], [20]. TOSSIM [18] is a simulator for TinyOS wireless sensor networks.

Prowler [19] is a MATLAB-based simulator that can simulate not only network protocol stacks but also radio transmission phenomena. Downey et al. [20] developed a flexible simulation framework, where a new model can be added or substituted easily. Note that the simulator used in this paper is to simulate the execution of a protocol based on our model.

8 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 a three step verification method that consists of nondeterministic analysis, probabilistic analysis, and simulation. Using this verification method, we can verify and analyze the correctness and performance of a sensor protocol specified in this state-based model.

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 setting of sensor networks.

The neighbor computation protocol in Section 5 is suitable for a resource limited sensor network, since each sensor needs to send a small number of messages to compute its strong neighbors. This protocol can be used to calibrate the model such that the estimated performance from the model is correlated to the observed performance from an actual prototype of the protocol.

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.

Acknowledgment

This work was supported by the Defense Advanced Research Projects Agency (DARPA) Contract F33615-01-C-1901.

References

1. Arora, A., Dutta, P., Bapat, S., Kulathumani, V., Zhang, H., Naik, V., Mittal, V., Cao, H., Demirbas, M., Gouda, M., Choi, Y., Herman, T., Kulkarni, S., Arumugam, U., Nesterenko, M., Vora, A., Miyashita, M.: A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. Computer Networks (Elsevier), Special Issue on Military Communications Systems and Technologies **46**(5) (2004) 605–634
2. Akyildiz, I.F., Su, W., Y.Sankarasubramaniam, Cayirci, E.: Wireless Sensor Networks: A Survey. Computer Networks, Elsevier Science **38**(4) (2002) 393–422

3. Mainwaring, A., Polastre, J., Culler, R., Anderson., J.: Wireless Sensor Networks for Habitat Monitoring. In: Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA (2002)
4. Liu, J., Chu, M., Liu, J., Reich, J., Zhao, F.: State-Centric Programming for Sensor-Actuator Network Systems. Pervasive Computing (2003) 50–62
5. Abdelzaher, T., Blum, B., Cao, Q., Chen, Y., Evans, D., George, J., George, S., Gu, L., He, T., Krishnamurthy, S., Luo, L., Son, S., Stankovic, J., Stoleru, R., Wood, A.: EnvioTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. In: Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan (2004)
6. Newton, R., Welsh, M.: Region Streams: Functional Macroprogramming for Sensor Networks. In: International Workshop on Data Management for Sensor Networks, DMSN (VLDB 2004). (2004)
7. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: a Tiny AGgregation service for ad-hoc sensor networks. ACM SIGOPS Operating Systems Review **36**(Winter) (2002) 131–146
8. Choi, Y., Gouda, M.G., Kim, M.C., Arora, A.: The Mote Connectivity Protocol. In: Proceedings of 12th International Conference on Computer Communications and Networks (ICCCN 2003), Dallas, TX (2003) 533–538
9. Wireless Embedded Systems. (http://webs.cs.berkeley.edu/)
10. Woo, A., Tony, T., Culler, D.: Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In: Proceedings of ACM SenSys, Los Angeles, CA (2003)
11. Cerpa, A., Busek, N., Estrin, D.: SCALE: A tool for Simple Connectivity Assessment in Lossy Environments. CENS Technical Report 21 (2003)
12. Gouda, M., Choi, Y.: A State-based Model of Sensor Protocols. Technical Report TR-05-46, Department of Computer Sciences, The University of Texas at Austin (2005)
13. Levis, P., Culler, D.: Maté: A Tiny Virtual Machine for Sensor Networks. In: International Conference on Architectural Support for Programming Languages and Operating Systems. (2002)
14. Gouda, M.G.: Elements of Network Protocol Design. John Wiley and Sons, Inc, New York, New York (1998)
15. Gracanin, D., Eltoweissy, M., Olariu, S., Wadaa, A.: On Modeling Wireless Sensor Networks. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004). (2004)
16. Volgyesi, P., Maroti, M., Dora, S., Osses, E., Ledeczi, A., Paka, T.: Software Composition and Verification for Sensor Networks. Science of Computer Programming (Elsevier) **56**(1-2) (2005) 191–210
17. Gouda, M., Choi, Y., Arora, A.: Antireplay Protocols for Sensor Networks. Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, (ed. Jie Wu), CRC (2005)
18. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In: Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003). (2003)
19. Simon, G., Volgyesi, P., Maroti, M., Ledeczi, A.: Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In: Proceedings of the IEEE Aerospace Conference. (2003)
20. Downey, P., Cardell-Oliver, R.: Evaluating the Impact of Limited Resource on Performance of Flooding in Wireless Sensor Networks. In: Proceedings of the International Conference on Dependable Systems and Networks, Florence (2004)