Redundancy-Aware Topology Management in Wireless Sensor Networks

Safwan Al-Omari and Weisong Shi
Wayne State University
{somari, weisong}@wayne.edu

Abstract— **Extending the lifetime of wireless sensor networks remains the most challenging and demanding requirement that impedes large-scale deployments. Studies show that considerable energy saving can be achieved only by putting a node's radio into full sleep mode. In this paper we present RAT, which is a redundancy-aware topology management protocol. RAT selects a minimum set of active nodes that are good enough to maintain connectivity, and allows others to sleep and save energy. RAT is designed and implemented with underlying wireless channel irregularity in mind. Scalability and low overhead are the other primary design goals of RAT as well. We implement RAT in the context of *Score*, which is a cross-layer framework that provides RAT with the neighbor set and allows RAT to coordinate its `SLEEP` and `ACTIVE` state changes with the routing layer smoothly. Using TinyOS and PowerTOSSIM, we implement RAT on top of *Score*. Comparing with the all-active scenario, RAT simulation results show a total energy consumption decrease of 67% in a one-to-many routing scenario and up to 87% in a many-to-one routing scenario.**

I. Introduction

Saving energy and extending the lifetime of unattended wireless sensor networks is still one of the most challenging design requirement of wireless sensor networks applications and protocols. To save energy and so extend the lifetime of wireless sensor network, researchers have considered power consumption at different levels, including the application level [8], routing layer [8], [12], and MAC layer [9], [21]. As previous studies showed [6], putting the node's radio into full sleep mode is the most efficient technique in saving the node's energy. Therefore, topology management protocols have the most potential in extending the network lifetime. In this paper, we use the term topology management to describe the distributed in-network process of selecting a set of active nodes, which together form a connected dominating set. All other redundant nodes can go to sleep and save energy.

Several topology management protocols have been proposed [5], [6], [14], [15], [16], most of these protocols presents a theoretical analysis of the connected dominating set problem [15], [16], while a few of them presents practical implementations of the problem [5], [6], [14]. Among the few, some suffer a high protocol overhead [3], which adversely affects the protocol's energy efficiency, while other have impractical assumption, which limits their applicability in real deployments.

In this paper we present RAT (**R**edundancy-**A**ware **T**opology Management), a novel topology management protocol to identify node communication redundancy (the *initialization phase*), and schedule nodes for sleep and active modes (the *scheduling phase*). RAT exhibits high fidelity to dynamic and irregular underlying wireless channels, while maintains low protocol overhead. Furthermore, RAT scales very well with high node densities as only active nodes are allowed to produce overhead traffic. To achieve high fidelity, RAT does not relate physical location to communication redundancy (e.g., fixed communication range), instead, it uses the neighbor set as the basis to define sensor node communication redundancy and to define a node's responsibility in the multi-hop network. To maintain low overhead, RAT leverages the neighbor discovery process and the broadcast nature of wireless links to establish communication redundancy knowledge among sensor nodes instead of asking nodes to simply exchange their neighbor sets.

RAT has two variants: **B**asic RAT (B-RAT) and **E**nhanced RAT (E-RAT). In B-RAT, the scheduling phase starts at the sink by triggering a scheduling thread, which after that propagates serially until all the nodes in the network switch to sleep or active modes. We trade some uniformity of active nodes distribution in the sensor field in E-RAT to overcome the potential scheduling thread deadlocks and to improve B-RAT propagation time. Instead of a single thread, E-RAT triggers several scheduling threads which propagate simultaneously in the network dispatching nodes sleep and active modes.

We implement RAT in the context of *Score* [1], which

is a cross-layer framework that allows different protocol layers to collaborate without the need for pair-wise interfaces. *Score* provides mechanisms to fill the gap between topology management and the other network services and protocols. This gap refers to the lack of mechanisms to communicate the sleep mode state to other protocol layers [5]. These mechanisms are important for the routing layer, e.g., to pro-actively build alternative routes. Also, *Score*, along with the proposed *neighbor discovery* service, provides an integrated and transparent neighbor set abstraction. In this abstraction, *Score* provides iterator-based interface to navigate, read, and update the neighbor records, while the *neighbor discovery* service uses passive and active listening, when necessary, to populate the neighbor set with the node's neighbors.

Using nesC [7] and PowerTOSSIM [13], we designed, implemented, and evaluated RAT and the *neighbor discovery* service using *Score* libraries and interfaces. For the evaluation we use two sets of performance metrics, *low level* and *high level*. We use the former to demonstrate attractive characteristics of the active nodes selected by RAT, such as the number of active nodes and how well these nodes are distributed in the sensor field. We use the latter to exhibit the advantage of *RAT* over the all-active scenario in energy savings using the two most popular communication patterns in wireless sensor networks; *one-to-many* and *many-to-one*. As we present in Section IV-B, RAT shows an energy savings of 67% in a *one-to-many* routing scenario and up to 87% in a *many-to-one* routing scenario for networks of high node densities.

The rest of the paper is organized as follows, Section II presents the RAT protocol and algorithm; Section III discusses implementation details including *Score* and the *neighbor discovery service*; In Section IV, we present our simulation setup and results. Related work and conclusion are presented in Sections V and VI respectively.

II. RAT Algorithm Design

RAT consists of two phases, the *initialization phase* and the *node scheduling phase*. In the initialization phase, each node becomes aware of its role (responsibility) in the multi-hop network, while in the scheduling phase, information from the initialization phase is used to put as many nodes as possible into sleep, while maintaining connectivity. Before delving into the protocol details, we present and explain key concepts and definitions which are important for the reader to follow up the discussion of RAT.

A. Basic Definitions and Notations

The neighbor set (NS_i) plays the central role in RAT and is used to define a node's responsibility in the multi-hop network and communication redundancy metric. Two nodes are considered to have a *high communication redundancy*, if they share a high percentage of nodes in their neighbor sets. A formal definition of *Degree of Communication Redundancy* of two nodes denoted as ($DoCR_{i,j}$) is presented next

$\underline{DoCR_{i,j}}$ describes quantitatively how much communication redundancy node j can provide to node i. Therefore, how much responsibility can node j take away from node i in the multi-hop network.

$$DoCR_{i,j} = \frac{|NS_i \cap NS_j|}{|NS_i| - 1}$$

Note that, $DoCR_{i,j}$ is not equal to $DoCR_{j,i}$ (i.e. asymmetric redundancy).

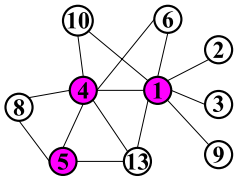
Based on the notion of the neighbor set, a node responsibility is defined as the number of nodes in its neighbor set. The more neighbors in a node's neighbor set, the more responsibility the node takes on, and so the more neighboring nodes are needed to provide the required communication redundancy if the node wants to switch to sleep. This intuition is formalized and quantified as the *neighbor set cover degree* of that node denoted as $CD_i(\alpha)$.

$\underline{CD_i(\alpha)}$ is defined as the minimum number of nodes in node's i neighbor set, which together can cover α portion of node's i neighbor set (NS_i), and is read as the α neighbor set cover degree.

From this definition we can see that α is an important parameter in calculating the *neighbor set cover degree* ($CD_i(\alpha)$), and directly affects the probability of having a disconnected network. We use α as a tuning parameter in RAT. α trades off energy saving to network connectivity. Choosing a low α value allows for higher levels of energy saving, but may result in a disconnected network. Choosing a high α value lowers the level of energy saving, but will more probably result in connected network. We use **T**hreshold of **C**onnectivity **C**onfidence (T_{cc}), denoted as (T_{cc}), as a representative value for α.

B. Example Scenario

Fig. 1 shows an example scenario of several nodes connected by wireless links as shown on the left side, from which we can find the neighbor sets (NS_i) and the pair-wise shared neighbor sets ($SNS_{i,j}$), which is the intersection of node i and node j neighbor sets, as shown in the central table. Based on this scenario, Fig. **??** shows



(a)

Node(i)	NS_i	d_i
1	{2,3,4,6,10,9,13}	7
4	{1,10,6,8,13,5}	6
5	{4,13,8}	3

$SNS_{(1,4)}$	{10,6,13}
$SNS_{(4,5)}$	{13,8}

(b)

Node(i)	DoCR$_{(i,j)}=\frac{\lvert SNS_{(i,j)}\rvert}{d_i-1}$	NSC$_i$(T$_{cc}$=80%)	CD$_i$(T$_{cc}$=80%)
1	DoCR$_{(1,4)}=\frac{3}{6}$	∞	-
4	DoCR$_{(4,1)}=\frac{3}{5}$	{1,5}	2
	DoCR$_{(4,5)}=\frac{2}{5}$		
5	DoCR$_{(5,4)}=\frac{2}{2}$	{4}	1

(c)

Fig. 1. An example scenario with a T$_{cc}$ value of 80%: (a) communication graph, (b) corresponding NS_i and d_i, and (c) DoCRi, j, NSC$_i$(T$_{cc}$) and corresponding CD$_i$(T$_{cc}$)

the pair-wise $DoCR_{i,j}$, $NSC_i(T_{cc})$, and $CD_i(T_{cc})$ for nodes (i = 1, 4, 5). The neighbor set cover of node i ($NSC_i(T_{cc})$) is simply the minimum set selected to calculate the *neighbor set cover degree* of node i ($CD_i(T_{cc})$). It is vital to note that no node maintains the entire tables, and the way these information is distributed over the nodes is presented and discussed next.

C. Redundancy-Aware Topology Management (RAT)

In the initialization phase all the nodes, which start from the `Active` mode, perform neighbor discovery (See Section III-B), in which nodes become aware of their neighbor sets and the pair-wise shared neighbor sets with each one of their neighbors. Nodes use the neighbor sets and the shared neighbor sets together to find the $DoCR_{i,j}$ and the $CD_i(T_{cc})$. B-RAT uses $DoCR_{i,j}$ to control the propagating scheduling thread, while E-RAT uses $CD_i(T_{cc})$ to trigger several scheduling threads in the network simultaneously.

1) Initialization Phase: The heart of the initialization phase is to obtain the pair-wise shared neighbor sets. A straight forward technique is for the neighbors to exchange their neighbor sets [6], but this can severely decrease the protocol's potential to save energy after all [3]. In RAT, we exploit neighbor discovery procedure and the broadcast underlying wireless communication to aid each node obtain the shared neighbor sets with each one of its neighbors efficiently. Technically speaking, obtaining shared neighbor set does not incur any new overhead traffic (The neighbor discovery has to be performed anyway). For now, we will assume that the neighbors are aware of their pair-wise shared neighbor sets, a detailed discussion of how we do this is deferred until Section III-B.

Once the neighbor set and the shared neighbor sets are available, each node i can locally calculate the $DoCR_{i,j}$ with each neighbor j and the $CD_i(T_{cc})$. Finding the $CD_i(T_{cc})$ is an NP-Complete problem (by simple reduction from subset sum), a greedy approximation is to order the nodes in the neighbor set according to their $DoCR_{i,j}$, and start including neighbor by neighbor (highest to lowest) in the neighbor set cover until the node reaches a coverage equal to or greater than the T_{cc} (Please, refer to [2] for detailed algorithm). By calculating the $DoCR_{i,j}$ list, and the $CD_i(T_{cc})$, node i is done with the initialization phase and ready for the scheduling phase, which is discussed next.

2) Scheduling Phase: In B-RAT, the sink (sender) triggers a scheduling thread by sending an active announcement. Recipient nodes initiate timers *proportional* to their $DoCR_{i,j}$ with the sender, so that nodes with least communication redundancy with the sender switch to the `active` mode first and so minimize the total number of active nodes in the network. All other nodes hold back their timers once they receive an active announcement. During the active announcements, any node that is able to collect enough neighbor set coverage becomes eligible for the `sleep` mode and switch to sleep immediately without sending any extra overhead messages.

B-RAT suffers a deadlock problem, which is possible when all the recipient nodes become eligible for the `sleep` mode leaving no node to pick up the scheduling thread. To overcome the deadlock problem in B-RAT, E-RAT uses the neighbor set cover degree to trigger scheduling threads. In a trial to minimize the total number of active nodes, E-RAT lets nodes with higher neighbor set cover degree switch to the `active` mode first and trigger a scheduling thread giving the opportunity for more nodes to collect enough neighbor set cover and switch to the `sleep` mode. The intuition is that nodes with higher neighbor set cover degree require more nodes to stay active and cover its neighbor set than a node with smaller neighbor set cover degree. In E-RAT, all nodes start in "E-RAT" operation mode by starting up timers *inversely proportional* to their $CD_i(T_{cc})$. If the timer fires before any scheduling thread reaches that node, the node switch to active and triggers a new scheduling thread to break a potential deadlock. On the other hand, if the node hears an active announcement before the timer fires, the node switches to normal "B-

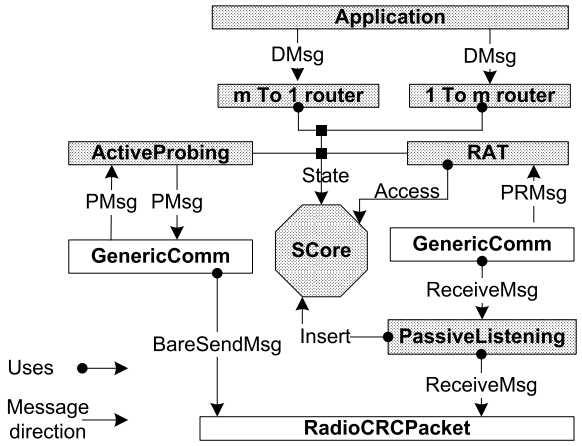


Fig. 2. Functional components: shaded boxes represent our modules, PMsg, PRMsg, and DMsg represent Probe, Probe reply, and Data messages respectively.

RAT" mode, in which later, the node can switch to active or sleep mode depending on how the existing scheduling threads propagate in the network, but the node will never trigger a new scheduling thread. If it happens that all B-RAT scheduling threads died, some node in the network will eventually break the deadlock by triggering a new scheduling thread as this node must have never been reached by any scheduling thread and still running in the E-RAT mode.

3) RAT protocol overhead: The RAT protocol overhead messages consists of two parts, overhead messages in the *initialization phase* and in the *scheduling phase*. By careful design of the neighbor discovery process, RAT can build the shared neighbor sets without introducing any new overhead messages, See Section III-C. To keep low overhead in the scheduling phase, only active nodes need to announce their states, as we present in Section IV, the number of active nodes stays constant even for higher density topologies, this ensures a constant message overhead during the scheduling phase.

III. Implementation Details

In this section we present the detailed implementation of RAT. Fig. 2 shows the functional decomposition including *Score* [1], *neighbor discovery* service, which consists of the *passive listening* and *active probing* modules, and finally the RAT module. The former two service modules provide *integrated* neighbor set abstraction and *neighbor discovery* service. This *integrated* service is used by RAT, which performs topology management. In the following section we briefly discuss *Score* interfaces, refer to [1] for more detailed discussion of *Score*.

```
interface SCore{
    // Sequential Access Iterator commands
    command result_t first();
    command result_t next();
    event result_t nextDone(uint16_t neighborID);

    // Ramdom Access Iterator command
    command result_t seek(uint16_t n_id);
    event result_t seekDone(result_t success);

    // SCore Reader
    command result_t read(uint8_t *neighbor);
    event result_t readDone(uint8_t *neighbor);

    // SCore Writer
    command result_t write(uint8_t *neighbor);
    event result_t writeDone(result_t result);
}
```

(a)

```
interface State{

   // To change the node's current state
   command result_t change(uint8_t newState)

   // Fired whenever the node's state changed
   event result_t changed(uint8_t newState);

}
```

(b)

Fig. 3. Score APIs, (a) neighbor set abstraction API and (b) state interface.

A. Sensor Core Module (Score)

Score sits at the core and provides other modules with three services. First, a unified neighbor set abstraction. Second, a modular cross-layer interface. Third, a cross-layer coordination mechanism.

1) Neighbor Set Abstraction API: Using *Score* access interface (Fig. 3(a)) a client module can read or write any neighbor record simply by pointing at the required record and performing a read or a write. Moving the pointer can be done in two ways, sequentially using the *first* and *next* commands, or randomly using the *seek* command (Fig. 3(a)). Following nesC/TinyOS philosophy, *Score* provides split-phase operations to keep the sensor responsive to external events. *Score* does not impose any limitations and is not involved in deciding which nodes are included in the neighbor set. In other words, *Score* only provides the mechanism and not the policy. The implementation of a specific neighbor policy is completely independent of *Score* and done by the *neighbor discovery* service.

2) Cross-layer Interface: To eliminate the need for pair-wise interfaces between the different network modules, *Score* defines a global neighbor record structure, in which each network module is allocated a number of bytes. A network module can use these bytes to annotate the neighbors with useful information that other modules wish to access. For example, A trust network service

can rank the neighbors based on some trust criteria, and annotate the neighbors with this value. Another service, the routing protocol for example, can access these trust values and exclude untrussed neighbors while building a routing tree.

To keep the *Score access* interface simple and general, *Score* does not provide individual read and write commands to read and write specific fields in the neighbor record, it only supports reading and writing entire records. By doing so, *Score* is not severely involved and dependent on a particular neighbor record structure, which we think can change in different implementations. Reading and writing entire records raises the need for *Score* to prevent network services unintentionally or intentionally (malicious service implementations) from overwriting each other's information in the neighbor record. Therefor, each network service is assigned a *writing mask*, This mask (for short) is statically defined in *Score* according to the current neighbor record structure. Each time a network service writes a neighbor record, *Score* will first apply the mask, on the new record, which sets all the unrelated bits to zeros, and then perform a bit-wise *and* operation with the old neighbor record. The masking process does not only provide inter-service overwrite protection, it also allows for multiple writers at the same time with no need for inter-service synchronization (each service writes its bytes only in the shared neighbor record).

3) Cross-layer Coordination: Score supports cross-layer coordination by maintaining a sensor node *operational state*, this *state* (e.g., `DISCOVERY`, `BOOTED`, `SLEEP`, and `ACTIVE`) describes the current sensor node operational status. Each network service can react in its own way when a new *state* is announced by *Score*. For example, The *active probing* module will send neighbor probing messages if the node state change to `DISCOVERY` (a `DISCOVERY` state means there are no enough neighbors in *Score*), while a routing service will hold its protocol messages as there are no enough neighbors to maintain a routing tree, and so save the precious node energy from being wasted for nothing.

Score provides a *state* interface (Shown in Fig. 3(b)), which provides a command to change the node current state and uses an event to announce state changes. Any network service wishes to react to state changes must provide implementation of the *changed* event, in which the service can take the appropriate action.

B. Neighbor Discovery Service

The *neighbor discovery* service works in *passive* as well as *active* mode. In *active* mode, the *neighbor discovery* service actively probes nodes in the vicinity to populate *Score* with new neighbors, while in *passive* mode, the *neighbor discovery* service passively intercepts incoming messages, extract the source address and insert a new record into *Score*. Other network services and protocols access the set of neighbors transparently using *Score access API* with no need for direct interface with the *neighbor discovery* service. *Active probing* module, Fig. 2, is responsible for active probing whenever *Score* announces a `DISCOVERY` node operational state, while *passive listening* module, Fig. 2, passively intercepts incoming messages and insert neighbor records into *Score*.

1) Active Probing: To perform active *neighbor discovery*, the *active probing* module simply broadcasts probe messages, which consists of the source node address (sender), nodes in the vicinity (receivers) reply by sending messages consisting of the the receiver's address and the original sender address. As we present in Section III-C, including original sender address in the probe reply messages is important to build the shared neighbor sets. The *active probing* module registers with *Score* for node operational state, whenever the operational state changes to `DISCOVERY`, *active probing* module starts probing for neighbors so that *passive listening* can insert new neighbor records into *Score*.Once the operational state changes back to `BOOTED`, the *active probing* holds back its probing messages.

2) Passive Listening: The *passive listening* module is placed under the *active messaging layer* so that it can intercept all incoming messaging and not only those targeted to the *neighbor discovery* service, and so lower the need for costly active *neighbor discovery*. To make this possible, the source node address has to be included as the first two bytes of the payload of all the packets, so that the *passive listening* module can simply extract the first two bytes of any incoming message and insert a new neighbor record into *Score*.

A ProbReply(i, j) message, on its own, tells any third party receiving node that both i and j are neighbors, RAT leverages this by overhearing these message to build and maintain the shared neighbor sets, a detailed discussion of this process is presented next.

C. Building Shared Neighbor Sets

As discussed above, all the nodes have to have access to their neighbor sets and the shared neighbor sets with

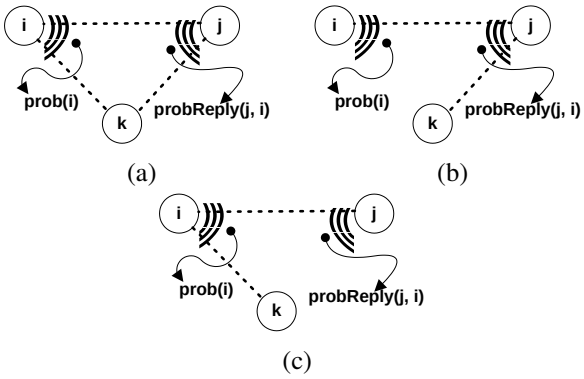


Fig. 4. Building shared neighbor sets at node k.

their neighbors to finish the initialization phase of RAT. The *neighbor discovery* process along with *Score* provide RAT with sequential as well as random access to the neighbor set. The shared neighbor sets are built and maintained by RAT itself. By overhearing a probe reply message sent from node j to node i, RAT at node k can add node j to the shared neighbor set of node i, and node i to the shared neighbor set of node j only if node i is in the neighbor set of node k. This scenario is illustrated in Fig. 4(a). Fig. 4(b) and Fig. 4(c) show the other two possible scenarios. In Fig. 4(b) node j and node k are neighbors, but node k and node i are not, in this case, node k will receive the probe reply message from node j, but never add any records to the shared neighbor sets as node i is not in node k neighbor set. In Fig. 4(c) node k and node i are neighbors, but node k and node j are not, in this case, node k does not receive the probe reply message from node j in the first place.

IV. Evaluation

A. Evaluation Setup and Metrics

Using nesC and PowerTOSSIM we implemented B-RAT and E-RAT and compared their performance to the all-active case. In B-RAT and E-RAT cases, only active nodes participate in the multi-hop network, while in the all-active case, all the nodes actively participate in the network and forward messages.

Two sets of performance metrics are used, the first is a low level metrics to show some attractive characteristics of the protocol, while the second set is a high level metrics to show the advantage of B-RAT and E-RAT over the all-active case in terms of energy savings. The first set includes four metrics. First is the *average active node degree*, which is the average of the number of active nodes in the neighbor sets of all the nodes in the network. Second is the *active degree distribution*, which is the number of nodes having the same active node degree. Third is the *sleeping ratio*, which is the ratio of sleeping nodes to the total number of nodes. Fourth is the *propagation time*, which is the time it takes the protocol (B-RAT and E-RAT) to finish and all the nodes switch to active or sleep.

In the second set, we use the total power consumption in comparing B-RAT and E-RAT to the all-active case. Two communication patterns are used, *one-to-many* and *many-to-one*, a detailed discussion of the experiments is presented later.

The experiments are conducted on five topologies. In all of the topologies, the nodes were randomly distributed over a fixed area of 100 by 100 units squared. In order to show the ability of B-RAT and E-RAT in leveraging high node densities, the average node degree in the topologies varies from seven to thirty, we use (Top 7, Top 10, Top 17, Top 21, and Top 30) to refer to them. The communication ranges are adapted to get the desired node degree, (Please refer to [2] for topology configuration details).

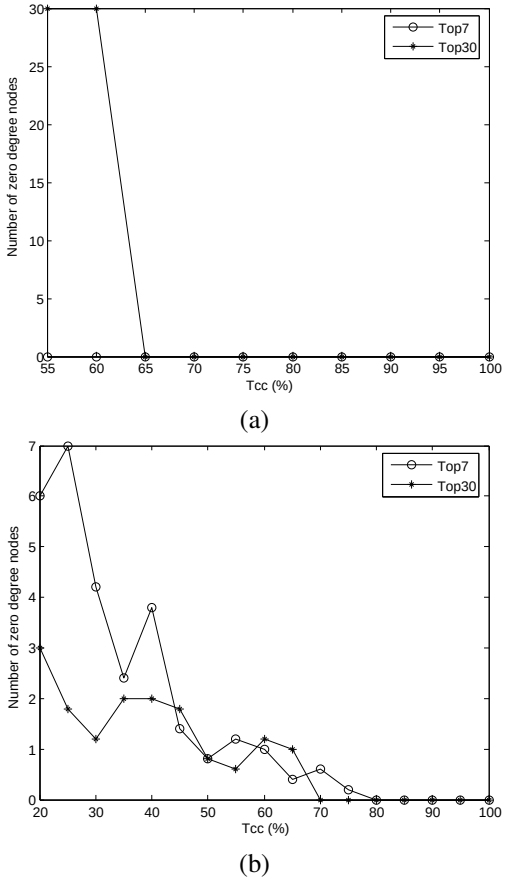


Fig. 5. Choosing appropriate T_{cc} values: (a) B-RAT, and (b) E-RAT value.

B. Simulation Results

In order to decide a good T_{cc} value, which is used to ensure a connected network, we conducted a set of experiments, in which we ran B-RAT and E-RAT using T_{cc} values ranging from 20% to 100%, each time the number of of zero degree nodes was recorded and plotted as in Fig. 5(a) and Fig. 5(b). The number of zero degree nodes is used as an approximation to find whether the network is connected. As the sleep mode eligibility is lenient for small T_{cc} values, B-RAT suffered deadlocks frequently, and in most cases the scheduling thread was not able to propagate over the entire network. Top 7 and Top 30 are used in these experiments as representatives, we can see from Fig. 5(b) that a T_{cc} value of 85% is good enough to ensure connectivity.

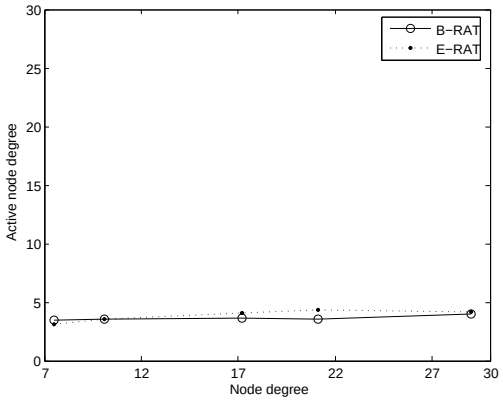


Fig. 6. The average active node degree of different topologies in B-RAT and E-RAT.

The second experiment shows the average active node degree after the application of B-RAT or E-RAT as the node density increases. A constant average node degree as the node density increases is important first to prove the correctness of the protocol and to show that the protocol actually has a constant overhead. The experiment was performed by running B-RAT and E-RAT on each one of the topologies and taking the average active node degree over all the nodes. Each value in Fig. 6 represents the average of 10 runs of B-RAT and E-RAT.

We can observe from Fig. 6 that both B-RAT and E-RAT maintains a constant active node degree as the deployment node density increases, which emphasizes two things: First, both B-RAT and E-RAT can leverage high node redundancy by selecting a constant number of nodes to be active which is only necessary for connectivity. Second, B-RAT and E-RAT maintain a constant protocol overhead (i.e. only active nodes announce their states in the protocol). In addition to the average active node degree, the active node degree distribution is important to show how uniformly the active nodes spread over the sensor field. Having a uniform active node degree helps in avoiding a situation where some network channels have a high contention (high node degrees), while others have low contention, which makes the decision of a MAC back off time for example difficult. An ideal distribution would be for all the nodes to have exactly the same active node degree, but this is impossible as boundary nodes by default have less node degree. Fig. 7 (next page) depicts the active node degree distribution for B-RAT, E-RAT, and the all-active case of three configuration, where the x-axis represents the active node degree values, while the y-axis represents the number of nodes with the corresponding active node degree. The figures show that in the all-active case, the node degree is highly variable. For example, in Fig. 7(c), some nodes have a degree of 40 while others have a degree of 10. In B-RAT and E-RAT, we can observe a neat distribution where most of the nodes have an active degree of 5.

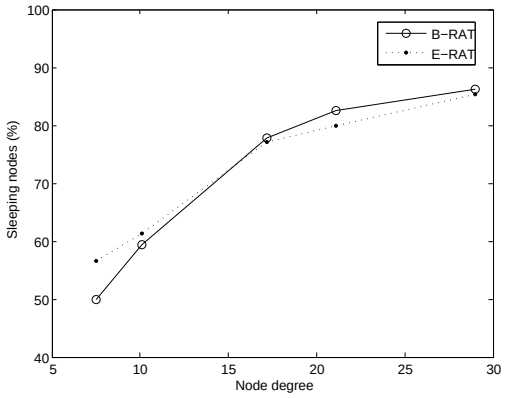
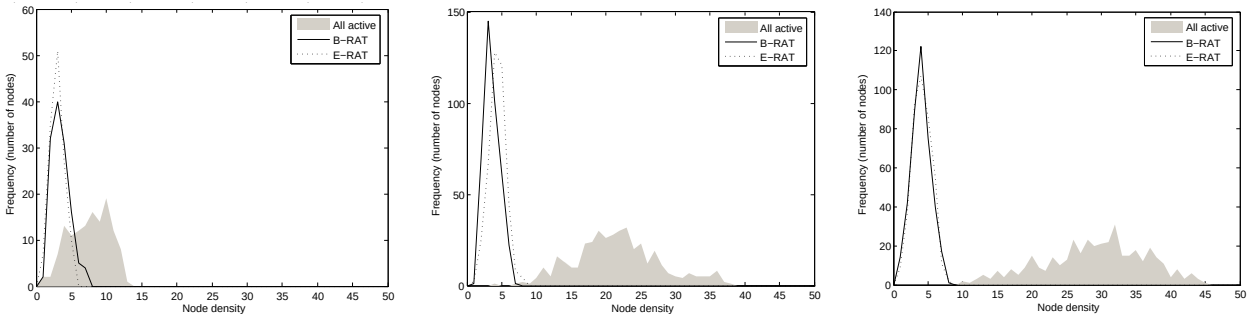


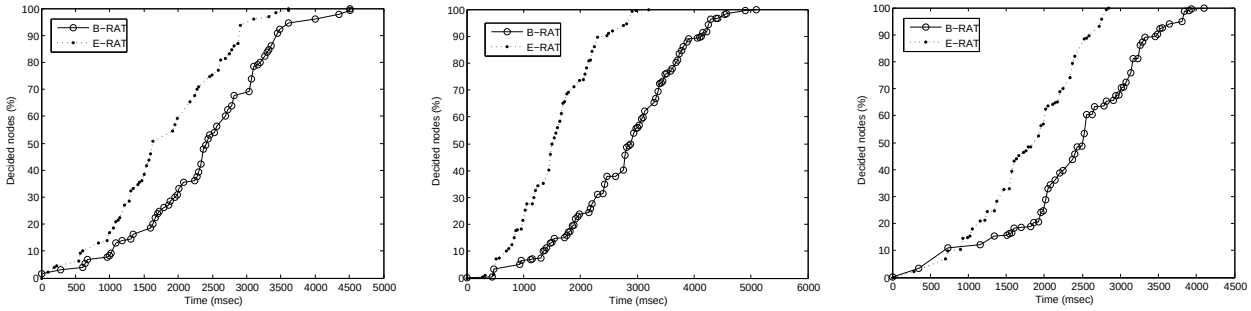
Fig. 9. Energy saving potential.

An understanding of the protocol's propagation and convergence time is important, other network services and protocols need to wait for the network to become stable before starting generating and sending data messages. Fig. 8 (next page) plots the cumulative distribution frequency of the decided nodes over time. By decided nodes we mean a node that either switches to `active` or `sleep` mode. Both can finish the protocol in less than 4 milliseconds. E-RAT can propagate faster in the network, which follows from the fact that E-RAT initiates multiple B-RAT threads concurrently in the network. In addition to a deadlock-free RAT, E-RAT has a faster propagation time.



(a) n=130 nodes. (b) n=330 nodes. (c) n=400 nodes.

Fig. 7. Active node degree distribution: (a) Top 7 (b) Top 21 (c) Top 30, n is the total number of nodes.



(a) (b) (c)

Fig. 8. The percentage of decided nodes vs time: (a) Top 7 (b) Top 21 (c) Top 30.

The ability of B-RAT or E-RAT in saving energy can be predicted from the percentage of nodes that switch to `sleep` mode. Fig. 9 shows an increasing percentage of sleeping node as the node redundancy increases, a percentage of more than 85% of the nodes switched to sleeping mode in topology 30.

In the last two experiments, we used the most two common communication patterns (one-to-many and many-to-one) to compare the performance of B-RAT and E-RAT to the all-active case in terms of the total power consumption. The sink in the *one-to-many* routing

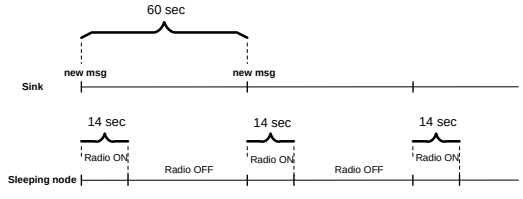
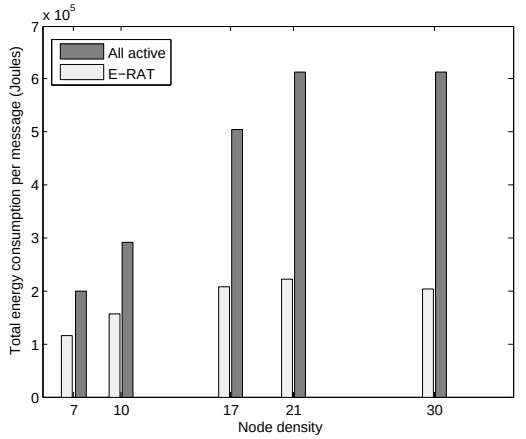


Fig. 10. Sleeping node duty cycle.

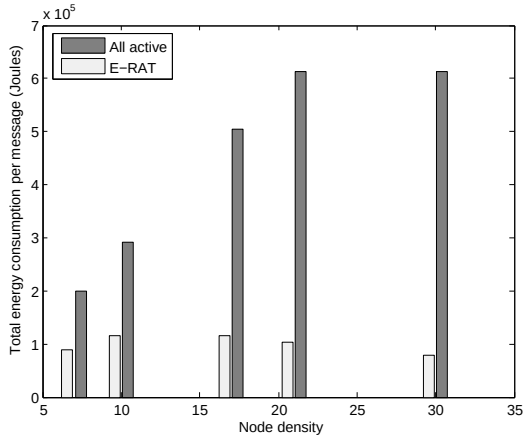
scenario, periodically (every 60 seconds) broadcasts a data message, and the recipient nodes rebroadcast again. To avoid forwarding the same data message more than once, each node forwards the same data message only once. In the all-active case, all the nodes stay active all the time and participate in the forwarding process, while in B-RAT and E-RAT, only active nodes stay active all the time and participate in message forwarding. On the other hand, sleeping nodes do not forward data messages and turn their radios off unless they need to receive data messages. A simple time line for the sink and a sleeping node is shown in Fig. 10. The figure shows that a sleeping node turn its radio on for 14 seconds starting from the time a new message generated at the sink, 14 seconds is the maximum time that a data message may take to propagate from the sink to the furthest node in the network. This time interval (i.e., 14 seconds) is a conservative value that accounts for the maximum one hop delay and the maximum number of hops in the network. A one hop delay may take up to 1 second, which is the maximum back off time a node may wait to avoid collisions, and the maximum hop in the topologies in our simulation is 14.

In the *many-to-one* routing scenario, the nodes in the network are arranged in a shortest path routing tree rooted at the sink, each node generates a data message every 60 seconds, which is forwarded up the tree after aggregated with other data messages to the sink. In the all-active case, all the nodes are active, join the routing tree, and participate in the data message forwarding,

while in B-RAT and E-RAT, only the active nodes form the routing tree and forward data messages, sleeping nodes turn their radios off unless they need to send a new data message generated locally at the node.



(a)



(b)

Fig. 11. Total energy consumption per data message: (a) One-to-many routing and (b) Many-to-one routing.

Fig. 11 compares the total energy consumption of E-RAT and the all-active case for three different topologies in the *one-to-many* routing scenario (Fig. 11(a)) and the *many-to-one* routing scenario (Fig. 11(b)). Since B-RAT and E-RAT show similar energy saving potential in Fig. 9 we use E-RAT only in our comparison with the all-active case. In both Fig. 11(a) and Fig. 11(b), the y-axis represents the total energy in joules consumed by all the nodes in the network for each individual data message generated in the network, while the x-axis represents the node density of each topology. Fig. 11(a) shows a total energy reduction of at least half in low density topologies (Top 7) and up to 67% in total energy reduction for higher density topologies (Top 30). Fig. 11(b) on the other hand, shows a total energy reduction of more than 80% for high node density (Top 30).

In the many-to-one routing scenario, Fig. 11(b), E-RAT shows an even increased energy saving over the all-active case compared to the *one-to-many* routing scenario, Fig. 11(a). This is due to the fact that in the *many-to-one* routing scenario in Fig. 11(b), sleeping nodes don't need to become active to receive data messages, instead, sleeping nodes become active only to send data messages, this allows for even lower duty-cycle of sleeping nodes.

V. Related Work and Discussion

Several topology management protocols have been presented in the literature [5], [6], [14], [20], RAT complements and enhances previous work by providing a working nesC/TinyOS implementation, which accounts for more practical issues such as irregular wireless channels. We also consider new important performance metrics in addition to energy saving such as active node density distribution and propagation time.

GAF [20] uses physical location to define communication equivalence and redundancy. The assumption that relates physical location to connectivity does not necessarily hold in real deployments [22], [23] and limits GAF applicability in real life deployments. Using the neighbor set to define communication redundancy, RAT adapts and captures harsh connectivity models. Exchanging long neighbor lists (high density deployments) among neighbors, as in SPAN and ReORG [6], [14] respectively, puts high overhead on the network. This high overhead limits the protocols' ability in saving energy after all. To avoid exchanging neighbor lists, RAT leverages the broadcast underlying medium of wireless channels to build communication redundancy information which is necessary for selecting the active node set. In addition to overhead resulted from exchanging neighbor lists, ReORG requires all the nodes to announce their states (active or sleep), this results in poor scalability with the total number of nodes. RAT on the other hand, requires only active nodes to announce their states, since the number of active nodes is constant relative to the total number of nodes, RAT scales very well to deployments with large number of nodes.

Also, the tight coupling of the routing and topology management layers, in SPAN [6], may un-necessarily limit the design space for the routing layer designers. RAT avoids any dependency on the routing layer, which makes it operational with any routing protocol, nevertheless the *state* interface provided by *Score* allows the routing layer to get notification of any topology changes

smoothly so that it can adapt its routing infrastructure appropriately. ASCENT [5] takes a different approach in selecting a set of active nodes, it uses the active node density and loss rate as driving factors in assigning nodes active and sleep states. Transient node failures and high wireless channel quality variation may compromise the integrity of a node's decision of going to sleep or active, which may lead to a disconnected network. RAT guarantees connectivity by forcing nodes to stay active unless a set of active nodes already provide enough communication redundancy.

Topology management, in which a set of nodes stay active while other nodes turn their radios off completely, is not the only mechanism that has been used to control network topology and so save energy. Several protocols and algorithms [4], [10], [11], [19] have been proposed to control network topology by adjusting the sensor node sending power. Such mechanisms can be used side by side to complement our work and further provide higher levels of energy savings.

Unlike previous neighborhood abstraction [17], [18], in which the goal was to support a unified neighbor view for application developers, *Score* is designed to support the system developers by providing neighbor set abstraction and mechanisms for cross layer interface and coordination.

VI. Conclusion and Future Work

We have designed and implemented *RAT*, *Score*, and *neighbor discovery* service, and evaluated RAT under two routing scenarios. RAT achieves energy savings up to 87% by leveraging high node density by assigning 80% of the nodes sleep mode and assigning a small set of nodes, good enough to maintain connectivity, active mode. Our future work is three fold. First, perform RAT continuously so that the topology is adjusted to environmental changes. Second, study the effect of RAT on the network capacity. Third, consider fault tolerance as another primary network property besides connectivity in RAT.

References

[1] S. Al-Omari, Junzhao Du, and W. Shi. Score: A sensor core framework for cross-layer design [extended abstract]. In *Proc. of QShine'06*, August 2006.

[2] S. Al-Omari and W. Shi. Redundancy-aware topology control in wireless sensor networks. Technical Report MIST-TR-2005-012, Wayne State University, November 2005.

[3] D. M. Blough and P. Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *Proc. of MobiCom'02*, September 2002.

[4] M. Cardei, J. Wu, and S. Yang. Topology control in ad hoc wireless networks using cooperative communication. *accepted to apear in IEEE Tran. on Mobile Computing*, 2006.

[5] A. Cerpa and D. Estrin. Ascent: Adaptive self-configuring sensor networks topologies. *IEEE Tran. on Mobile Computing Special Issue on Mission-Oriented Sensor Networks*, July 2004.

[6] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. SPAN: An energy-efficient coordination algorithm for topology maintenance in ad-hoc wireless networks. In *Proc. of MobiCom'01*, July 2001.

[7] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proc. of PLDI'03*, June 2003.

[8] B. Hamdaoui and P. Ramanathan. *Energy-Efficient and MAC-Aware Routing for Data Aggregation in Sensor Networks*. IEEE Press, October 2004.

[9] S. Jayashree, B. S. Manoj, and C. S. R. Murthy. On using battery state for medium access control in ad hoc wireless networks. In *Proc. MobiCom'04*, September 2004.

[10] N. Li, C. Hou, and L. Sha. Design and analysis of an mst-based topology control algorithm. In *Proc. of INFOCOM'03*, March 2003.

[11] N. Li and J. C. Hou. Localized topology control algorithms for heterogeneous wireless networks. *IEEE/ACM Trans. on Networking*, December 2005.

[12] R. Shah and J. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Proc. of WCNC'02*, March 2002.

[13] V. Shnayder et al. Simulating the power consumption of large-scale sensor network applications. In *Proc. of ACM SenSys 2004*, November 2004.

[14] M. Yarvis W. S. Conner, J. Chhabra and L. Krishnamurthy. Experimental evaluation of synchronization and topology control for in-building sensor network applications. In *Proc. of WSNA'03*, September 2003.

[15] P. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, April 2004.

[16] Y. Wang, W. Wang, and X. Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *Proc. of MobiHoc'05*, May 2005.

[17] M. Welsh and G. Mainland. Programming sensor network using abstract regions. In *Proc. of NSDI'04*, March 2004.

[18] K. Whitehouse, C. Sharp, D. Culler, and E. Brewer. Hood: A neighborhood abstraction for sensor networks. In *Proc. of MobiSys'04*, June 2004.

[19] J. Wu and F. Dai. Mobility-sensitive topology control in mobile ad hoc networks. *accepted to apear in IEEE Tran. on Parallel and Distributed Systems*, June 2006.

[20] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proc. of MobiCom'01*, July 2001.

[21] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proc. of Infocom'02*, June 2002.

[22] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proc. of SenSys'03*, November 2003.

[23] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proc. of MobiSys'04*, June 2004.