IMPLEMENTATION OF TEAN-SLEEP FOR WIRELESS SENSOR NETWORKS

Jesus Jaquez, David Valencia, Manikanden Balakrishnan,
Eric E. Johnson and Hong Huang
New Mexico State University
Las Cruces, NM
August 19, 2009

ABSTRACT

Wireless Sensor Network have many potential applications, including environmental monitoring and threat surveillance, among many others. Sensor nodes are limited in processing capabilities, radio range, storage but most importantly in energy. When a node's energy is exhausted, the node can no longer provide sensor data to the network, nor can it forward data from other nodes. This can result in a network partition. One way a node may extend its lifetime is by the use of sleep. In many of the applications of WSNs the nodes are idle most of the time until an event occurs and information needs to be transmitted. A node is able to save energy while idle by sleeping. Topology and Energy Adaptive, Non-synchronous (TEAN) sleep provides the nodes in a network a mechanism to save energy by sleeping while also keeping a connected network. In this paper, we present an implementation of TEAN-sleep in tinyOS on a 50 node test bed of Crossbow TelosB motes. Preliminary test results show significant improvement in network lifetime when TEAN-sleep is engaged, with insignificant degradation of connectivity.

I. Introduction

Wireless Sensor Networks consist of sensor nodes which have the capability to communicate, process information, and sense the area around them. These devices work together in order to perform such tasks as environmental and disaster monitoring [1].

In many applications nodes are idle for a huge percentage of their lifetime. Many approaches have been used in order to try to extend the operational lifetime of a network. These include topology control and energy harvesting, among many others.

In this paper we provide an implementation of the TEAN-sleep algorithm [2] in tiny Operating System (tinyOS) [3]. This implementation is tested on Crossbow Telosb motes [4]. The paper gives a review of TEAN-sleep, details regarding its implementation, performance analysis, and finally conclusions.

A. Related Work and Original Contributions

Conserving energy and extending the lifetime of sensor nodes is an important topic in the area of WSNs. Many proposals have been made in order to address possible solutions to these problems. Researchers have investigated the use of sleep scheduling as in [5] and some which pair sleep scheduling with the reduction of redundant sensing by nodes which cover the same area [6]. In [7] locally available information is used in order to make decisions about sleeping. Each node calculates a drowsiness factor, an estimate of a sensors current energy. It then will compute its Shout Time Delay. If the node has a small Shout Time Delay then it is more likely to be able to sleep.

Original Contributions The original work [2] aimed at introducing the framework, and simulations were done under idealistic channel conditions. The predicted alpha-performance of connectivity based sleep coordination is heavily dependent on the underlying link behavior, and thus analysis under real wireless channel environments is required to quantify TEAN-sleep performance. In that perspective, the following are original contributions of this paper:

- Update of Sleep Eligibility Condition
- Implementation and verification of TEAN-sleep algorithm
- Link establishment and maintenance
- Design and implementation of the routing layer

II. TEANSLEEP

TEAN-sleep was first introduced in [2]. The premise of TEAN-sleep is to allow nodes in a network, who are idle, to sleep, given that they meet certain energy and connectivity requirements, which are described in the following sections. TEAN-sleep has the following characteristics [2]:

- Topology adaptive: WSNs are densely deployed for robustness against frequent node and link failures, which implies only a subset of nodes is required to establish a connection backbone for reliable information supply. The rest of the nodes could sleep to conserve energy without impacting network connectivity. TEAN-sleep applies a sleep condition that ensures network connectivity is maintained and only the redundant nodes are switched off. Nodes adapt their sleep decisions based on locally computed neighborhood topology.
- Energy adaptive: When preparing to sleep, a node will calculate the time it can go to sleep based of its energy level relative to the average energy of its 1-hop neighbors. This will cause energy starved nodes to sleep for longer amounts of time.
  Sleep Duration, T_S,is calculated as:

$$T_S = \frac{1 - P_A}{P_A} T_A \tag{1}$$

Equation 1 defines the amount of time that a node will be able to sleep. In this is equation P_A is the probability that a

node is in an active state. The formulation of P_A is described in equation 2. T_A is the average amount of active time within a duty cycle.

$$P_A = \bar{P_A} - \frac{\bar{E} - E_i}{\lambda_A T_{converge}} \quad (2)$$

$\bar{P_A}$ is the average probability that your neighbors are in an active state. $\bar{E}$ is the average of your neighbors energy levels. E_i is a nodes current energy level (Joules). λ_A is equal to the dissipation rate (J/s) in active state. $T_{converge}$ is the amount of time in which a node should try to converge to the neighborhood energy average.

In this work $\bar{P_A}$ has been set to 0.5. Our intent is that every node have a duty cycle of 50%. λ_A is a constant set to the dissapation rate when a TelosB mote is in active state with its radio on. Finally $T_{converge}$ has been set to 3 Hello Intervals (HI). As given in [2] we should see about a 50% duty cycle. A 50% duty cycle may seem to be high for a sensor network. It has been chosen in order to shorten the length of testing this implementation.

Based on these equations we can see that as a node drops below its neighborhood average it will begin to sleep longer. Conversely it will sleep less if its current energy level is greater than the average of all of its neighbors.

- Free sleep: The amount of sleep is not governed by any fixed duty cycle. In TEAN-sleep a duty cycle is defined as a listening period followed by a sleep period. A node who goes to sleep is never woken up by some predetermined duty cycle.
- Distributed, minimal overhead: Sleep decisions are made by a node and based on information a node has gained about the network from its 1-hop neighbors. The message exchanges incur some energy overhead but it is mitigated by sleep.

A. Control Messages

There are two types of control messages in TEAN-sleep. They are the Hello message and the Sleep beacon. These messages are used in order to exchange information with 1-hop neighbors. Node identifications are not used in either message payload because they appear in the the tinyOS packet header and can be retrieved from it.

Hello messages are broadcasted periodically, every Hello Interval (HI). They include information about a nodes current 1-hop neighbors and the amount of energy it currently has. This information provides each node with a 2-hop view of the network and is used when calculating the Sleep Eligibility Condition (SEC). Figure 1 shows the structure of a Hello message.

The maximum payload of a packet in tinyOS is 28 bytes. This allows for listing of up to 26 1-hop neighbors In the future an implementation of packet fragmentation algorithm may allow inclusion of more neighbors.

A sleep beacon is transmitted when a node determines that it is eligible to sleep. This packet contains the amount of time

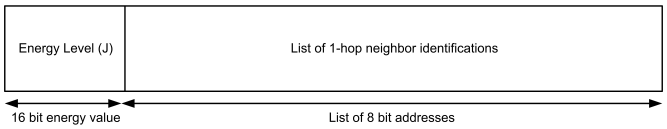

Fig. 1. Hello Packet

that a node is going to be asleep. It also serves as a way to keep 1-hop neighbors informed about a nodes current status. Upon the reception of a sleep beacon a node will recompute his SEC to reflect this change. The Sleep beacon is shown in Figure 2.

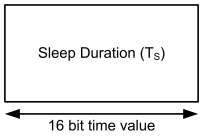

Fig. 2. Sleep_beacon Packet

B. Network Information Table (NIT)

As stated above, A node will store information from Hello packets and Sleep beacons of all 1-hop neighbors. The NIT is shown in table I. It contains a list of all 1-hop neighbors, all neighbors of 1-hop neighbors (extended neighbors), current sleep state, and expiry time. The SEC is based upon this information.

TABLE I
NETWORK INFORMATION TABLE

1-hop neighbor	sleep flag	Extended Neighbors	expiry time
Neigh 1	Yes	list of 1's neighbors	3*HI
Neigh 2	No	list of 1's neighbors	3*HI
...	...	...	...
Neigh n	No	list of n's neighbors	3*HI

C. Sleep Eligibility Condition (SEC) and Update

The original sleep eligibility condition states the following: "A node, n, is eligible to sleep if and only if all 1-hop neighbors in n's NIT, both awake and sleeping, are connected to at least one other currently awake neighbor of n."

This condition seems very intuitive but in the process of implementing this work we have found a case in which this condition fails. Figure 3, shows an example in which the above definition fails. In this example node 5 should be unable to sleep, since doing so causes a partition in the network. Nodes 6, 7, 8, and 9 are cut off from the rest of the network.

It can be seen that all of node 5's 1-hop neighbors (3,4,6, and 7) are connected to another of node 5's 1-hop neighbors: nodes 3 and 4 are mutually connected as are nodes 6 and 7. Node 5 would compute an SEC equal to 1. Due to this we have adapted the SEC based upon a nodes hop count to the sink.

Updated SEC: "A node, n, is eligible to sleep if and only if all 1-hop neighbors, both awake and asleep, with a hop count

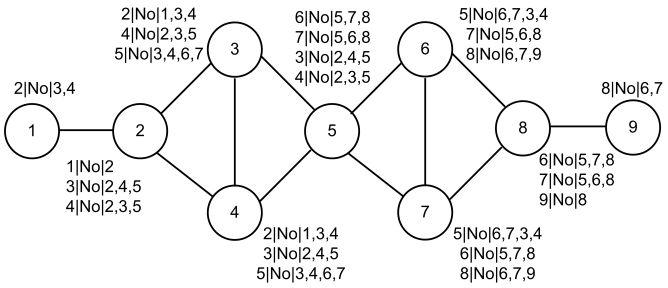

Fig. 3. SEC Failure Example

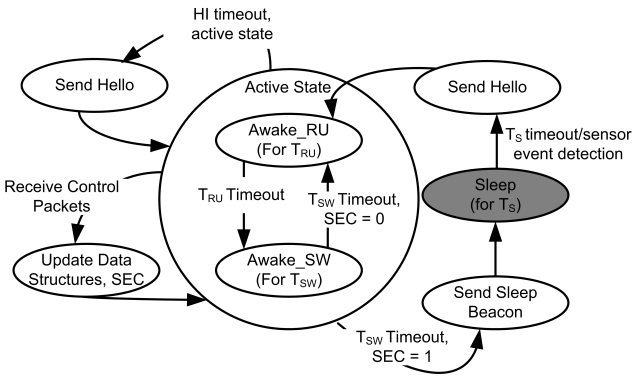

Fig. 4. TEAN-sleep State Machine

greater than n's hop count is connected to at least one other currently awake 1-hop neighbor of n with a hop count less than or equal to n's hop count."

Now returning to figure 3, node 5 would compute a SEC equal to 0. This is due to the fact that two of its neighbors, 6 and 7, are not connected to any other nodes with a hop count less than or equal to that of node 5.

The updated SEC assumes that the routing being used only forwards packets based upon hop count and it will never send packets to a neighbor whose hop count is equal to or greater than that of the originating node. This routing scheme is used in this implementation and given in more detail later in this paper. Although this updated SEC will provide us with a more reliable condition for sleeping, it also limits the use of a byzantine routing protocol.

D. Energy Model

The energy model used in this implementation is based off measurements given for the TelosB mote given by Crossbow. All nodes start with a set energy level. It is assumed that new batteries have the same amount of energy.

A node updates its current energy level after sending or receiving any control message or whenever a node has a timeout for the Hello packet, i.e. every HI. Sending and receiving account for when a node is active, whereas the timeout for Hello packets accounts for times when a node is idle but has its radio on. Table II shows the power consumption of the TelosB mote obtained from Crossbow [4].

TABLE II
POWER CONSUMPTION MODEL

Mote Operating Mode	**Power Consumption (mW)** (with 3V supply)
Active, Radio Transmit	57.6
Active, Radio Receive	74.4
Idle(Active, Radio Idle	0.063
Sleep(Radio/MCU off)	0.018

E. State Machine

TEAN-sleep can be viewed as a state machine, figure 4, with the following states:

- Awake_RecordUpdate (Awake_RU) serves to provide nodes with time to receive control packets and update their NIT. They stay in this state for a time of T_{RU}, which is equal to one HI.
- Awake_SleepWait (Awake_SW) is used to randomize the instances at which nodes will go to sleep if they are eligible. A node will stay in this state for a time of T_{SW}. Equation 3 stipulates the amount of time that a node will be in this state. If a nodes probability of being awake, P_A (equation 2), is high then it will stay in this state longer than a node with a low P_A.

$$T_{SW} = P_A \frac{HI}{2} \tag{3}$$

- Sleep is the state in which a node has calculated that it meets all the requirements to power down its radio and put the microprocessor in the lowest possible power state. The amount of time the node is in the sleep state is the sleep duration, T_S. This value is energy adaptive with respect to a nodes energy level and the average energy level of its 1-hop neighbors. The sensors on the nodes are not shut down. This provides, in the case of event monitoring, a means to wake a node up if the application deems it necessary due to some event.

Upon bootup nodes employing TEAN-sleep will send out a Hello Message. This message serves as a signal that a node is actively participating in TEAN-sleep. The node then goes into the first state of the state machine, Awake$_{RU}$. In this state a node actively listens for any updates from its 1-hop neighbors. After a time of T_{RU} the node then transitions into the $Awake_{SW}$ state. In this state a node calculates its timeout T_{SW}. A node will then check if it is eligible to sleep.

If the node is eligible to sleep it will send out a sleep beacon to inform its 1-hop neighbors and then go to sleep for T_S. Once it awakens it sends a Hello message and repeats the process. Conversely, if a node is ineligible to sleep then it returns to the Awake$_{RU}$ state and repeats the process. Nodes actively send out Hello packets every HI while in an active state. For more information on TEAN-sleep please refer to [2].

III. IMPLEMENTATION

In order to test TEAN-sleep, network construction and a routing layer needed to be implemented. They are described in

the subsequent sections. These two layers were kept as simple as possible while still providing adequate performance to the system. It was also decided that these would be developed locally to ease their integration with TEAN-sleep.

A. Network Construction

Network construction occurs in the following fashion. Every node in the network periodically sends a ping message, which announces that it is looking for neighbors. All nodes which can hear this message, i.e. 1-hop neighbors, will respond with a pong message. Upon reception of the pong message the originating node will append an entry to its route table for each source of a pong message. These links are marked as active and bidirectional. Each responding node will wait for an acknowledgment to be received; if the acknowledgment is received, an entry is appended to its route table and the entry is marked as active and bidirectional, otherwise it will be set to active and unidirectional. Figure 5 shows the format of a ping message.

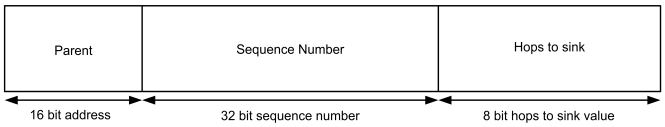

Fig. 5. Ping Message

The ping message contains information about the nodes parent and its hop count to the sink. The pong message contains information about its parent, its hop count, and also a Received Signal Strength Indicator (RSSI) measurement of the ping packet. The Pong message format is shown in Figure 6.

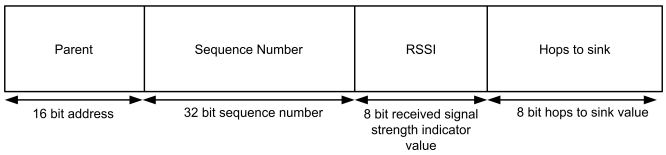

Fig. 6. Pong message

Network discovery begins at the sink node and expands outwards as nodes send ping messages. Furthermore, network discovery is periodic. Nodes update their route tables upon reception of either message. This also gives a basis for the number of hops in the network and is a shortest path problem.

Initially a node reports a hop count of 255. This number describes that the node currently has no route to the sink. It also reports a parent of 0, this informs neighbors that the node does not have a valid parent.

Once a node adds an active link it must receive another ping from the node within 3 ping periods in order to keep the link active. This is done to avoid stale links in the routing table. If a link becomes stale it is set to inactive and and all values (hop count, parent, etc) are reset. This is a precautionary measure so a node does not use stale information when choosing a parent.

At startup the sink is the only node which has a hop count, set to 0. It reports this value. All of the 1-hop nodes will report a hop count of 1. Their 1-hop neighbors, who are not also neighbors of the sink, will now report a hop count of 2. This process propagates from the sink outward to the other nodes in the network.

B. Routing

The routing algorithm used is proactive and uses hops to the sink, obtained from ping messages, to determine which node it will route to. Nodes maintain a route table that is updated upon the reception of any ping or pong message as well as any control message from TEAN-sleep.

When a node needs to deliver a packet to the sink it will first find its neighbor with the lowest hop count to the sink. It will then check if the link is still active. If the link is still active it will check to make sure that it is not the parent of the node it is going to send to. This precaution occurs in order to counter routing loops. It also makes sure that the node has a valid parent. Finally, if TEAN-sleep is being used, it will make sure the node is not currently asleep. If all these conditions are valid it will select this node as the destination of its packet. Once received at the destination node, it will go through the same procedure. This will happen until the data packet makes its way to the sink.

C. Forwarding Data

Packets are sent using a forwarding engine. The forwarding engine uses the pool component that is provided in tinyOS [3]. This component is a dynamic memory pool and is used to buffer data messages upon their reception or creation. In this work two memory pools are used. One of the pools is used for data messages generated locally (source message pool) and the other pool is used for data messages which were generated at other nodes (forward message pool).

It also uses the queue component that is provided in tinyOS. It is a first in first out (FIFO) queue and accepts elements that are pointers to the messages in the pools.

Packets generated locally and from other nodes in the network will be passed to the forwarding engine. The forwarding engine will then be put the messages into the respective pool. The packet will then be put into the queue and then sent. Once sent, the packets will be dequeued and released from the appropriate pool.

D. Application

The application being tested is based on uniform events, which is similar to a typical surveillance scenario. The data packet structure, Figure 7, and its Header, Figure 8, are shown below. Currently, data generation occurs every minute from the time a node boots. A data packet is generated by populating the packet values from the following sensors: Voltage, Photo Radiation, Solar Radiation, Temperature, and finally attaches the energy level the node currently has. This packet is then passed to the forwarder and handled accordingly.

The header contains a source field, a sequence number and also a time to live (ttl) value. The source is the node

identification number of the node which originated the packet. The sequence number is the current packet number, all nodes initially start with a sequence number of 0 and it is incremented every time a packet is generated. Finally a packet has a max number of hops to reach the sink, the ttl value; if this number is exceeded the packet is dropped.

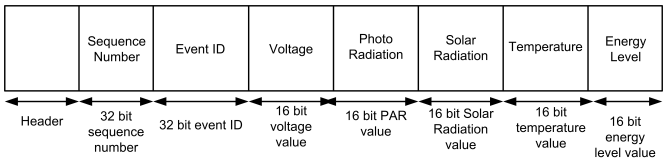

Fig. 7. Data Packet

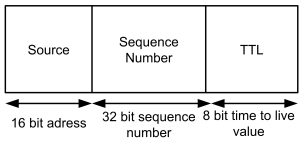

Fig. 8. Header of Data Packet

E. Architecture

This implementation was separated into several components. Separation of components was done in order to restrict the access between different components in the system and more importantly for clarity. The implementation of this algorithm is encapsulated under one file, the configuration file, which provides all components to the application layer. The components consist of the following:

- Route Table - a generic module which provides access to a data structure which maintains 1-hop neighbor information
- Routing Engine and Network Construction - provides the logic for network discovery, link maintenance, and finally the selection of neighbors to forward data to
- Forwarding Engine - provides access to data forwarding
- TEAN-sleep - provides the implementation of TEAN-sleep, described herein

When booted a node first starts the application. Once the application has started it will start the routing engine. The routing engine in turn starts TEAN-sleep and the forwarding engine. A state variable is used in order to keep the current state of each component (on or off).

The components act as they were described above. If a node calculates that it is eligible to sleep it will signal the application layer. The application will then turn off the routing engine. The routing engine will then turn off the forwarding engine. The TEAN-sleep component is never shut down when entering the sleep state. This is because the call to sleep is triggered in the TEAN-sleep component. To wake up the node a timer is set with a time of T_S. Once a timeout occurs TEAN-sleep will signal the application that it should wake up the other components. The application restarts the routing engine, which restarts the forwarding engine.

If the TEAN-sleep service needs to be shut down by the application it can be turned off. This may need to happen if say an event, such as a volcanic eruption, occurs and packet delivery supersedes network lifetime.

IV. Experimental Setup

TEAN-sleep was tested in the Electrical and Computer Engineering building at New Mexico State University. Node 1, the sink, is set up in the middle of a hallway on the second floor of Thomas and Brown. It is connected to a computer for logging of incoming data packets. A second node placed next to the sink is sniffing packets from the network. It is not actively participating in this experiment. On the second floor of the building we have 21 additional nodes. Eight nodes, 3 in the east stairwell, 3 in the west stairwell, and 2 nodes in the south stairwell are bridge nodes to the remaining 21 nodes which are placed on the third floor of Thomas and Brown Hall. Figure 9 shows a representation of the second and third floor layouts used for this experiment, nodes within the dashed boxes are the stairwell nodes.

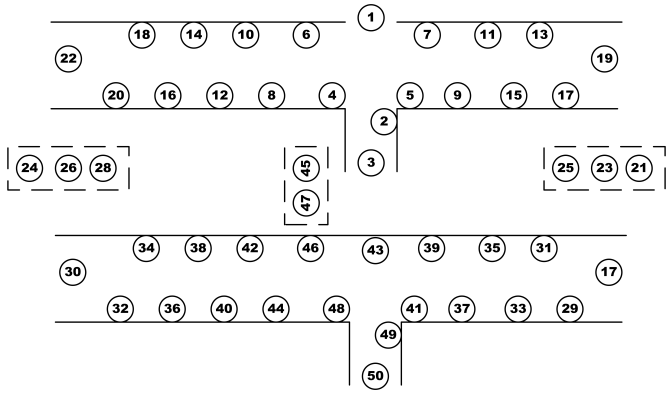

Fig. 9. Node Placement, second floor (top) and third floor (bottom)

V. Performance Analysis

In this section we present an analytical model, as well as the results from our experiments. The analytical model is presented in order to give a prediction of lifetime for nodes at a lower active duty cycle than the one used in the experiment.

A. Analytical Model

In this analytical model we have made several assumptions. It is assumed that every node behaves perfectly. Nodes always have an awake period followed by a sleep period. Another assumption made is that each node has ten neighbors in its route table. Forwarding of data is handled by assuming that each node is uniformly distributed and that each hop area contains 10 nodes. This model also did not account for any sleep provided in tinyOS.

Equation 4 gives the model for the lifetime of a node. $L(x)$ is the measure of lifetime in seconds. x is the active portion of the duty cycle and is bounded between 0 and 1. E_{usable} is the usable energy of two AA batteries and $D(x)$ is the dissipation rate and is defined in equation 5.

$$L(x) = \frac{E_{usable}}{D(x)} \quad (4)$$

The dissipation rate consists of two terms. The first being $A(x)$, it is the power dissipated while a node is in an active state. The second, $S(1-x)$, is the power dissipated while a node is in a sleep state.

$$D(x) = A(x) + S(1-x) \quad (5)$$

In order to calculate $A(x)$ a snapshot of the network was taken. This snapshot encompassed every packet, transmitted and received and accounts for times when the node is idle. $S(1-x)$ is given in the telosB data sheet [4]. In order to find E_{usable} a power curve was integrated using Simpson's Rule.

In this analytical model we have made calculations for node lifetime depending on hop count to the sink. Figure 10 shows the predicted lifetime for a node at a one-hop distance to the sink. Since a one-hop area will forward a higher rate of packets to the sink than any other region the network should become partitioned, and therefore unavailable once the 1-hop area has died. The analytical model predicts that a node not using sleep will last 5.5 days. With a 50 percent duty cycle a node will stay alive for 6.05 days and finally at a 25 percent duty cycle a node will die after 12.34 days.

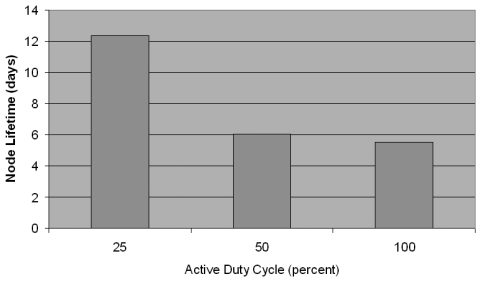

Fig. 10. Analytical Model: 25,50, and 100 percent duty cycles

In this model, a node must use an active duty cycle of no more than 54 percent. At this point and above the amount of overhead incurred by TEAN-sleep is detrimental to lifetime. In order to get a node lifetime of months or years we would need to set the active portion of the duty cycle to about 10 percent or lower. At a 10 percent active duty cycle the lifetime, acording to the model, for a node would be approximately 30 days. Clearly droping the active duty cycle closer to 0 percent would have the effect of expanding lifetime.

B. Results

Experiments were run according to section IV. Nodes are started up with the evens from 2 through 50 in ascending order and then the odds 3 through 49 in descending order.

Figure 11 shows the results from the experiment. The network, for experiments which did not use sleep, was available for a total of 6.96 Days. In the TEAN-sleep tests the network was available for 9.63 days. In these tests the network was deemed available as long as as 85 percent of the nodes in the network were still reporting data packets to the sink.

Obviously a network which employs sleep will have a longer lifetime than one that does not. That being said, these results show that sleep coordination can be handled at the neighborhood level. Another concern was how sleep would interact with the routing layer. In these experiments there was no significant degradation in the reports from nodes further away from the sink.

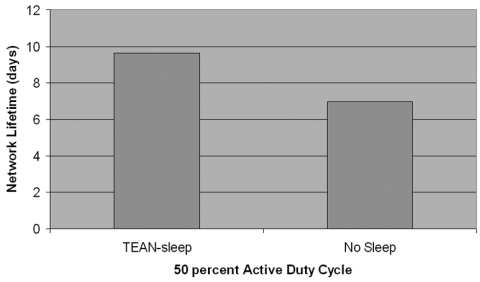

Fig. 11. Experimental Results for 50 percent duty cycle

As stated earlier the model used herein did not account for any sleep that was provided by tinyOS. This may account for some of the error between the theoretical value and the observed values. It was decided not to disable any sleep patterns offered by tinyOS as it would skew the performance of TEAN-sleep. In an actual deployment of nodes one would never disable any energy saving techniques.

VI. Conclusions and Future Work

This paper provided the details of an implementation of TEAN-sleep in tinyOS, which was tested on Crossbow TelosB motes. It showed that sleep coordination efforts can be done at the neighborhood level and showed the savings in energy inherent with sleeping.

Testing TEAN-sleep under varying duty-cycles is of great importance. It is important in order to validate the impact of TEAN-sleep on the performance of links as well as other layers, particularly the routing layer.

Nodes in a high density network will have a greater chance of sleeping, thus extending the lifetime of a network. It would be of great advantage to look at the performance of TEAN-sleep with varying node densities.

In this work we paired TEAN-sleep with simple yet adequate network construction and routing algorithms TEAN-sleep would benefit from more robust algorithms. The SEC condition used in this work is based on hop count. It would extend the operation of TEAN-sleep if it were able to route packets in any direction. For this to happen a new SEC condition would need to be created and validated.

References

[1] F. Zhao and L. Guibas, *Wireless Sensor Networks: An Information and Processing Approach*. Morgan Kauffman Publishers, 2004.

[2] M. Balakrishnan, E. E. Johnson, and H. Huang, "Tean-sleep for distributed sensor networks: Introduction and alpha-metrics analysis," in *Proc. IEEE Military Communications Conference MILCOM 2007*, 29–31 Oct. 2007, pp. 1–7.
[3] [Online]. Available: http://www.tinyos.net/
[4] Http://www.xbow.com/. [Online]. Available: http://www.xbow.com/
[5] D. Shuman and M. Liu, "Optimal sleep scheduling for a wireless sensor network node," in *Proc. Fortieth Asilomar Conference on Signals, Systems and Computers ACSSC '06*, Oct. 29 2006–Nov. 1 2006, pp. 1337–1341.
[6] S. Chachra and M. Marefat, "Distributed algorithms for sleep scheduling in wireless sensor networks," in *Proc. IEEE International Conference on Robotics and Automation ICRA 2006*, 15–19 May 2006, pp. 3101–3107.
[7] G. Simon, M. Molnar, L. Gonczy, and B. Cousin, "Dependable k-coverage algorithms for sensor networks," in *Proc. IEEE Instrumentation and Measurement Technology*, 1–3 May 2007, pp. 1–6.