To appear in WISARD'09, a workshop in IEEE COMSNETS 2009

REEF : A Reliable and Energy Efficient Framework for Wireless Sensor Networks

Arun Kumar K.S
Naval Physical and Oceanographic Laboratory
Kochi-682021, INDIA
Email: arunks1978@gmail.com

Vinay Joseph Ribeiro
Dept. of Computer Science and Engineering
IIT, ND-110016, INDIA
Email: vinay@cse.iitd.ernet.in

***Abstract*—This paper presents an efficient and scalable framework called Reliable and Energy Efficient Framework (REEF) for reliable data collection in Wireless Sensor Networks. REEF employs a distributed scheme which enables it to scale to large networks. It partitions the network in to clusters where the node with highest residual energy in a neighborhood become the cluster head. REEF forms a virtual backbone connecting the cluster heads and the sink by selecting some nodes from each cluster as gateway nodes. Sensor nodes report sensed data to their respective cluster heads which use an outlier detection algorithm to detect faulty data. REEF significantly cuts down on energy consumption by ensuring that a large number of sensor nodes can go into a deep sleep mode, in which the radio as well as CPU are switched off, for a major part of their life time. Simulation results demonstrate that REEF uses as low as 50% of the energy for the same accuracy when compared to a recently proposed scheme based on passive listening.**

I. Introduction

A Wireless Sensor Network typically consists of a large number of nodes which are equipped with a low end microprocessor, a radio for communication and a number of sensors depending on the physical phenomena that are to be measured [1]. These sensor nodes help to monitor natural phenomena like temperature, humidity etc. accurately in space and time or detects events of interest [2]. A few special nodes called *sinks* gathers data collected by various nodes in the network. Sinks normally are more powerful than the ordinary nodes in terms of processing power, available storage and available energy. A large number of ordinary (non-sink) nodes collect data and send the collected data to these sinks. A non-sink node can perform the following tasks - collect and process data and use a radio to communicate with other nodes. The processing capabilities of ordinary nodes are limited and high end processing is carried out at the sinks. Sinks can also act as gateways to other networks like the Internet.

The sensor nodes are normally small in size and the energy available for each node is limited. Each node, once it is deployed, operates unattended till its battery drains. Accessing a sensor node for battery replacement may be impossible due to the hazardous nature of the environment where the nodes are deployed.

Sensor nodes may be implemented using inexpensive hardware and may not be highly reliable. This results in erroneous readings reported by some of the sensor nodes. The faulty values reported by the sensor nodes have to be filtered out to improve integrity and reliability of the sensor network. This is of utmost importance particularly in the case of sensor networks which employ aggregation techniques, because the result of an aggregation can drastically change because of a fault that was present in the data set used for aggregation.

This paper presents a framework called REEF (Reliable and Energy Efficient Framework) for improving the reliability of wireless sensor networks by detecting faulty values reported by sensor nodes without compromising energy efficiency. REEF forms clusters and runs an outlier detection algorithm on each cluster head to detect faulty data in readings reported by member nodes. Each member node can sleep for most of the time since it need not passively listen to check if the sensed reading matches with those reported by its neighbor nodes. Also, the faulty readings are filtered out at the cluster head level and no energy is spent in communicating the faulty readings to the sink. Hence the energy consumption in the network reduces significantly and network life time and reliability increases.

The rest of the paper is organized as follows. In Section II we give an overview of related work. After describing REEF in detail in Section III we compare it with another related scheme using simulations in Section IV. We list our conclusions and suggest future work in Section V.

II. Related Work

A brief overview of the existing fault models, clustering algorithms and schemes for improving reliability in Wireless Sensor Networks are provided in this section.

A. Fault Models

Faulty data reports from a sensor node may be either due to fault in its sensor or due to the fact that the node is captured by an adversary and reprogrammed to give faulty data in order to make the network ineffective. This paper deals with the case where faults are unintentional and not due to any malicious nodes captured by adversaries.

The kinds of faults observed in real world sensor deployments and the models for those faults are presented in [3], [4], [5] and [6]. Let y be the true value of a physical phenomenon. The value read by a non faulty sensor is given by

$$x = y + \epsilon$$

where $\epsilon \sim N(0, \sigma_m^2)$ is the measurement noise which is Gaussian. When the sensor is faulty, the value read takes the general form

$$\tilde{x} = \beta_0 + \beta_1 y + \epsilon_f$$

This formula is the generalization of following specific kinds of faults.

Short Fault or Spike Fault: Such a fault results in a sharp change in the measured value between two successive data points. The faulty reading is modeled as

$$\tilde{x} = \beta_0 + y + \epsilon$$

Variance Degradation Fault: The variance degradation fault occurs when the sensors become less accurate over course of time. If actual measurement variance is σ_m^2, the fault variance is σ_f^2, where $\sigma_f^2 \gg \sigma_m^2$. The sensor noise is $\epsilon_f \sim N(0, \sigma_f^2)$. The value read by faulty sensor will be

$$\tilde{x} = y + \epsilon_f$$

Stuck-At Fault: This fault represents the sensor getting stuck at a particular value. Most often this value is the lower or higher end of sensing range. This is the most common type of fault observed in sensor nodes.

$$\tilde{x} = \beta_0$$

B. Clustering Algorithms

Clustering is widely used in Wireless Sensor Network to reduce energy consumption as well as to increase scalability. The nodes in the network are grouped in clusters and a node from each cluster is selected as the cluster head. Each node in the network will either become a cluster head or join one of the clusters as a member. Instead of sending data reports directly to the sink, the member nodes transmit their data packets to their respective cluster heads. Cluster heads are responsible for in-network processing like data aggregation and data fusion [7], [8]. A number of clustering algorithms have been proposed in the literature and design of efficient clustering algorithm has become a major topic in wireless sensor network research.

Low Energy Adaptive Clustering Hierarchy (LEACH) presented by Heinzelman et al. forms clusters by using a distributed algorithm, where nodes make autonomous decisions without any centralized control [9]. A node decides to be a Cluster Head (CH) with a probability *p* and a non-CH node determines its cluster by choosing the CH that can be reached using the least communication energy.

Younis et al. present the Hybrid Energy Efficient Distributed (HEED) protocol for clustering [10]. A node is selected as a cluster head depending on whether other cluster heads are its one hop neighbors and its own residual energy. Cluster heads can communicate with each other using more transmission power than that used for intra-cluster communication. HEED integrated with a multi-hop routing scheme provided in TinyOS is described in [11] as iHEED.

Distributed Clustering Algorithm (DCA) for Ad Hoc Networks is presented by Basagni et al. in [12]. The cluster head is selected using a weight-based criteria and a node decides its own role solely knowing the information of its one hop neighbor. The main idea behind DCA is that a node decides which role to assume only when all its neighbors with bigger weights have decided their own roles.

C. Schemes for Improving Reliability

A number of methods for improving reliability and fault tolerance using outlier detection in Wireless Sensor Networks have been proposed by researchers. A model based error correction scheme which is based on the temporal correlation of sensor data is presented by Mukhopadhyay et al. in [13]. The model is built offline and is used for on-line prediction and error correction of sensor readings.

Zhang et al. gives a method to pinpoint exactly the compromised nodes in a sensor network [14] . In this method, each sensor node generates an alert when it observes a neighbor reporting a reading that is inconsistent with its own reading. These alerts are received by the base station which in turn build a trust graph and assign each node faulty/non faulty status such that this assignment doesn't affect the consistency of the graph.

Chen et al. present a distributed algorithm which runs in phases for fault detection [15]. In the first phase, each sensor compares its value with reading of its neighbors and declares itself *locally good* if its value matches with the majority of its neighbors. In the second phase, those nodes which are *locally good* and have a majority of their matching neighbors also *locally good* are declared *globally good*. In the last phase, neighbors of *globally*

good nodes which do not agree with those nodes are marked *faulty*.

Balzano et al. discuss the usage of a Beta Reputation System for improving reliability in wireless sensor networks [16]. In this scheme, each sensor node runs a watchdog module and reputation module. The watchdog spies on neighboring nodes' data reports and then run some outlier detection methods to assign a level of confidence for each of the neighbor nodes. The reputation module takes the output of the watchdog (direct reputation) as well as reputation tables transmitted by neighboring nodes (indirect reputation) to calculate reputation and trust values for each of the neighbors. This scheme is distributed and scalable. But there is overhead due to periodic exchange of reputation values between the nodes and due to the continuous promiscuous listening used by all the nodes.

Zhuang et al. presents a wavelet based scheme for in-network outlier cleaning in Wireless Sensor Networks [17]. The time series data from each sensor, consisting of sensor readings over a given time period, is used by the sink to determine similarities between time series data sent by neighboring nodes.

A framework for secure aggregation using neighborhood watch is presented by Rabinovich et al. in [18]. Several methods like mutual monitoring and constraint validation, randomized delivery trees and redundancy in data supplied by the network are used to make the network resilient to attack. The Beta reputation system to detect and exclude compromised nodes.

III. Architecture of REEF

REEF uses spatial correlation in the readings of sensors deployed in a neighborhood to detect outliers (faulty readings reported by sensor nodes). The deployed sensor nodes are grouped in to clusters. A node in each cluster is selected as a cluster head. The readings from each sensor node in a cluster is transmitted to the cluster head. The cluster head then uses an outlier detection algorithm to detect outliers in the spatially correlated readings.

A. Clustering Algorithm

The algorithms discussed in Section II make assumptions which may not be always true for WSNs. LEACH assumes that all nodes are within communication range of each other and the base station. The authors of LEACH suggest the use of multihop routing to relax this constraint. They however do not present details of such a routing protocol in the case where the majority of the nodes are sleeping (in order to save energy) [9]. The clusters formed are not well distributed in space and the cluster head may not always be a one hop neighbor of the member nodes if realistic range of communication is assumed.

HEED selects cluster heads on the basis of residual energy and uses an increased transmission power (four times the power used for intra cluster communication) for inter cluster communication [10]. But increased transmission power may not always guarantee that all neighboring cluster heads can be reached since there may be physical obstacles to radio waves. The attenuation of signal may not always follow the inverse square law so that increasing power by four times does not always make it possible to reach a node two hops away. Although iHEED integrates HEED with a multihop routing protocol, it doesn't consider residual energy of nodes while making the routing decision [11].

An assumption used in DCA is that a message sent by a node is received correctly within a finite time by *all* its neighbors which is not realistic assumption for the wireless medium. It also doesn't explain how to create a virtual backbone connecting the cluster heads. None of these algorithms takes in to account the existence of asymmetric links which may be present in the wireless networks. They assume that if a node A can hear node B and that node A added node B to its neighbor list, then node B must have also added node A to its neighbor list. These algorithms also do not take care of conditions when cluster control messages may be lost due to collisions.

The algorithm used in REEF is an extension of the Distributed Clustering Algorithm (DCA). It is based on more realistic assumptions and take in to consideration the asymmetric links and lossy links that may exist in the real world deployment of wireless sensor networks. It operates in phases and uses only local information about one hop neighborhood at each node to decide if that node should function as a cluster head, gateway node or an ordinary cluster member. The time for which each node will execute a particular phase is preprogrammed and the transition of nodes from one phase to another take place in a synchronized fashion.

The *Hello Phase* is used for neighbor discovery. In this phase, the router of each node is configured to disable routing and forwarding and the nodes periodically transmit *Hello packets*. The *Hello packet* contains the residual energy of a node, a list of neighbors already discovered and a number of packets successfully received from each of the neighbors. When a node receives a *Hello packet* from another node, it updates its *Neighbor Table* with the information available in *Hello packet*. At the end of this phase, each node has a list of neighbors, their residual energy and the information about the fraction of total packets successfully received from and sent across to each neighbor stored in its neighbor table.

The *Cluster Setup Phase* follows the hello phase. The neighbors with receive rate and send rate less than a predefined threshold are removed from *Neighbor Table*.

A node after entering the clustering phase selects itself as a Cluster Head and periodically transmits a *Cluster Head packet* if it is the node having the highest residual energy among its one hop neighbors. If a node receives a *Cluster Head packet* from a node having highest residual energy in its neighborhood, it joins that cluster by sending a *Join packet*. The details of clustering algorithm is given in [12]. The *Cluster Head packet* and *Join packet* are transmitted periodically by the cluster heads and member nodes respectively till the clustering phase is over. At the end of the clustering phase, if a node has not decided which cluster it should join because it is waiting for the status of some higher energy node in the neighborhood or has not received any acknowledgment from the cluster head (in the form of including its ID in the list of members in the *Cluster Head packet*), that node elects itself as a Cluster Head.

All the nodes enter the *Scheduling Phase* following the Cluster Setup phase. This phase is used by Cluster Head nodes to transmit a TDMA schedule for transmission to the member nodes through *Schedule Packet.* The member nodes use this phase to inform the respective cluster heads to function as gateway nodes. Those nodes which have at least one node from another cluster as neighbor are called boundary nodes. We require that link between a boundary node and a neighbor node in a different cluster should have a low packet error probability in both the directions. This requirement is ensured by the fact that two nodes list each other in their *Neighbor Tables* only if the packet error probability in both directions is low. The *Gateway Candidature* packet is sent periodically by the boundary nodes.

The *Steady Phase* follows the scheduling phase. In the initial part of this phase, the cluster head nodes select the gateway nodes in their clusters which can connect to neighboring clusters. There can be 3 cases

- The cluster head node itself has a node belonging to another cluster as a neighbor
- A node in the cluster has another cluster head as its neighbor
- A node in the cluster has a non cluster head node in another cluster as neighbor.

For each neighboring cluster, it selects the boundary node with maximum residual energy among those that can connect to the neighbor cluster to be a gateway node. It should be noted that when the boundary node cannot connect to a neighbor cluster head directly, minimum of the residual energy of the boundary node and its neighbor in the foreign cluster is used as a basis of selection. The cluster head nodes transmit the *Gateway Selection packet* periodically for a small amount of time (which is preprogrammed). A member node select itself as a gateway if its ID is included in the list of gateways in the *Gateway Selection packet* send by its cluster head or it has at least one neighbor whose membership status (whether a cluster head or cluster member) is unknown.

The router functionality in each of the cluster heads and gateways are enabled. The nodes which are neither cluster heads nor gateways switch off their radios, sensors and CPU and enter deep sleep mode where only a clock which can awake the CPU after a prescribed time period is running. These nodes wake up periodically (depending on their TDMA slot position obtained from *Schedule packet*), sense the environment and sends the data read to the respective cluster head. The gateways also send data to their cluster head according to the TDMA schedule received. But they never enter deep sleep mode. Shallow sleep is executed by the cluster heads and gateway nodes when the CPU is idle and no radio signals are detected. Any radio activity switches the nodes which are in shallow sleep mode to the active mode. The cluster heads and gateway nodes actively participate in routing and forms the virtual back bone connecting all cluster heads to the sink. The cluster head periodically perform outlier detection and data fusion on the data received from member nodes and send the fused data packet to the sink. Each cluster head also transmits the data collected during each round of data collection to the neighboring cluster heads through the gateway nodes in the form of *Cluster Head Data packet*. This is done so that more data is available at each cluster head for doing outlier detection.

The *Steady Phase* is followed by *Extended Steady Phase* in which the nodes which are neither cluster heads nor gateways do not transmit any data packets. This phase is included to guarantee that fused data packets containing the data collected during the last round of data collection in the steady phase reaches the sink before all the nodes enters the hello phase. The absence of this phase may result in some fused data packet getting discarded en route to the sink because routing is disabled in hello phase. All the nodes enters hello phase after the extended steady phase. It may be noted that a node enters hello phase if and only if its residual energy is greater than a predefined threshold (a small percentage of the initial energy). If the residual energy is low, the node will not execute the hello, cluster setup and schedule phases of clustering algorithm. In steady phase, it enables the routing functionality and routes its data to the sink via one of the neighboring cluster head or gateway nodes.

B. Outlier Detection

A number of schemes proposed in literature for outlier detection and improving reliability were discussed in Section II. Some of these schemes are model based [13]. Such schemes require a large quantity of measured data to be available a priori for developing an accurate model offline. Some others are centralized and do not scale up

well [14].

The Scheme using Beta Reputation System [16] requires all nodes to be in promiscuous listening mode and there is periodic exchange of reputation values between the nodes. As a result, this scheme is not energy efficient.

REEF uses a distributed approach which is scalable and energy efficient. Each cluster head runs an outlier detection algorithm to detect outliers among readings reported by its members.

A good outlier detection algorithm should detect most of the faults and the number of false positives must be small. REEF uses "Spatial Averaging Algorithm" for outlier detection. This algorithm uses the median which is statistically robust to outliers [19]. It is rule based and hence doesn't require a comparison with the estimated standard deviations (which are affected by presence of outliers) of readings to decide whether a value is an outlier or not.

For each node in the cluster, the median of the readings of neighbor nodes is calculated. If the reading of the node differs from the median by more than a threshold value, it is declared as an outlier. The algorithms is displayed in listing 1. It is assumed that the mean and standard deviation of the measurement error (calibration error) of the sensor used on board is provided by the manufacturer. The threshold is taken as twice the maximum measurement error.

Algorithm 1: Spatial Averaging Algorithm

```
Input: Readings of all the nodes in a cluster
Output: Outlier factor for each node
foreach node i do
    Sort the readings of the neighbors ;
    med = median of readings ;
    if abs(reading-med) > threshold then
        OF_i = 1.0
    else
        OF_i = 0.0
    end
end
```

IV. Simulations and Results

The performance of REEF was compared with the Reputation based Framework for Sensor Networks (RFSN) presented in [16]. We choose RFSN for the comparison because it is similar to REEF in many ways. Like REEF, RFSN is not model-based, it exploits spatial correlation in measured data, and it is distributed (and hence scalable). In contrast to REEF, however, RFSN employs promiscuous listening on all nodes in the sensor network. We use the J-Sim simulator for simulating REEF and RFSN.

In RFSN, each node runs a Local Outlier Factor (LOF) outlier detection algorithm on the data gathered by passive listening and subsequently calculates reputation for each of its neighbors. Typically, it takes some time before the reputation of a misbehaving node in RFSN goes down. We found through simulations that this slow tracking actually reduces performance particularly in case of short faults. We hence use the output of the LOF algorithm directly while comparing performance of RFSN and REEF and leave out computation of reputation. This also allows a more fair comparison with REEF which is not reputation-based. We modified RFSN so that the sink computes the outlier factor for a node i from the LOF values assigned to that node by its neighbors as follows.

$$OF_i = \frac{abs(LOF_i - \mu_{LOF})}{\sigma_{LOF}} \quad (1)$$

where μ_{LOF} and σ_{LOF} respectively are the arithmetic mean and standard deviation of LOF values. The sink considers a node faulty if the outlier factor is greater than a *threshold*.

In REEF, the cluster heads transmit the outlier factor of each node to the sink along with the different readings and corresponding time stamps.[1] The outlier factor received at the sink is used to determine if the value reported by a node is reliable or not. REEF uses the "Spatial Averaging" algorithm (see Algorithm 1) for outlier detection. The outlier factor assigned for each reading in this algorithm is either 0 or 1. The sink treats a node as faulty if its outlier factor is 1.

We compare the number of detected faults and false positives for both the algorithms for different topologies. In addition the energy spent by the algorithms and the network life are also compared.

A. Experimental Setup

In the simulations, the nodes are deployed in a field of area 100m x 100m. The number of nodes in the network was varied from 25 to 200. Topologies were generated randomly. The sink was placed in the upper-left corner of the sensor field. The data to be reported by each node was stored in separate files. The data file for a node contains the data values that are to be reported and the time at which each data value is to be reported. Spike Fault, Stuck-At Fault and Variance Degradation faults were simulated. The simulation parameters and their values are given in Table I.

Since it is important to use realistic values for the CPU processing time during the simulation, both the algorithms were implemented to run on java-based SunSPOT

[1]The sink can itself compute all the outlier factors because it has access to data readings from all nodes. This centralized solution is, however, less scalable than REEF's distributed one. Note that centralized solutions cannot be used for applications performing in-network data aggregation. Although REEF is ideally suited for such applications, we do not consider them in our simulations in order to facilitate a fair comparison with RFSN which does not perform in-network data aggregation.

Simulation Parameter	Value
Number of nodes	25,50, 100 and 200
Size of the sensor field	100m x 100m
Battery Capacity	720 mAH
Battery Voltage	3.7 V
Initial Energy	9590 J
Radio Range	30m
Routing Algorithm	AODV
MAC layer algorithm	CSMA
Current drawn by Radio in Tx mode	18 mA
Current drawn by Radio in Rx mode	20 mA
Current drawn by Radio in idle mode	426 μA
Current drawn by Radio when Off	1 μA
Current drawn by CPU in Active Mode	80 mA
Current drawn by CPU in shallow sleep	31 mA
Current drawn by CPU in deep sleep	33 μA
Current drawn by Active Sensors	6mA
Current drawn by Sensors when Off	5 μA
Data reporting periodicity for RFSN	300 sec
Data reporting periodicity for REEF	300 sec
Periodicity of Setup packets in REEF	Random : 1 to 30 sec
Duration of Setup phases in REEF	300 sec
Duration of Steady phase in REEF	3600 sec

TABLE I
SIMULATION PARAMETERS AND VALUES

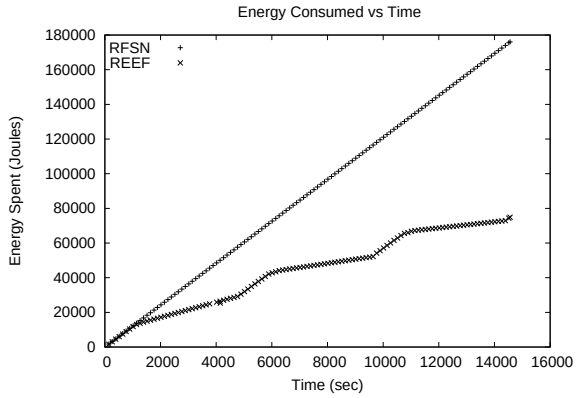


Fig. 1. Energy vs. Time for 100 nodes

motes and the processing time taken by different key procedures in the algorithms were measured. For procedures whose time of execution depends on the number of neighbors or number of members, the number of neighbors and number of members was assumed to be 10. REEF and RFSN were simulated for different topologies by varying the number of nodes and position at which the nodes are placed. For a network of a given size, ten different topologies and thirty different data sets (ten data sets for each fault) were used. Simulation was done for all topology-data set combinations. The results of the simulation for a network of size 100 nodes are given in following subsections. Results for networks with size 25, 50 and 200 nodes were found to be similar.

1) Energy Spent in the Network vs. Time: Figure 1 shows the energy spent in a network of 100 nodes as time advances. The energy spent by RFSN is more than

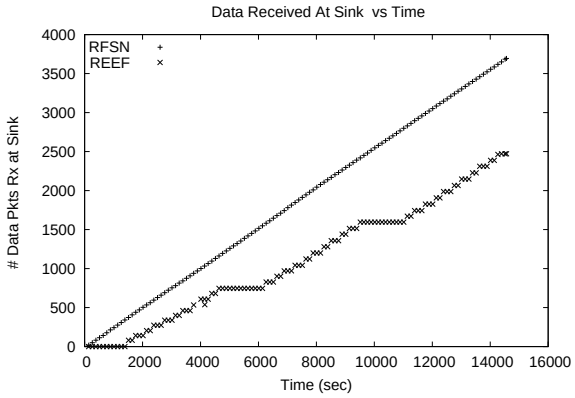


Fig. 2. Data Received at Sink vs. Time for 100 nodes

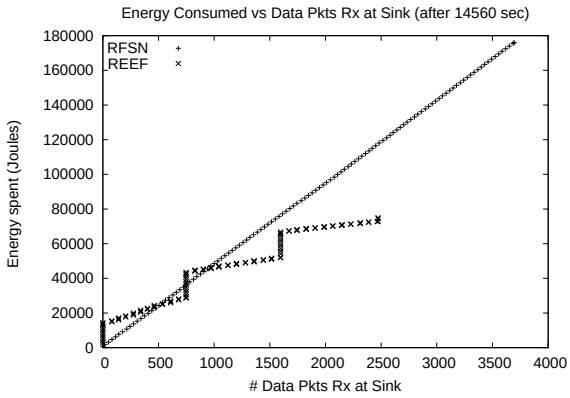


Fig. 3. Energy Spent vs. Data Received at Sink for 100 nodes

that spent by REEF since in RFSN nodes use passive listening and the never move to deep sleep state where radio and CPU is switched off. In addition, all nodes participate in routing. Observe that the energy spent increases linearly with time.

In REEF, the nodes which are not gateways or cluster heads can make use of deep sleep mode. They need to wake up periodically to sense the environment and send the data to cluster head. Only cluster heads and gateways have their router functionality enabled. We note that the energy spent in steady phase is less than the energy spent in setup phases.

2) Data Received by the Sink vs. Time: A comparison of the number of data packets received by the sink for a network of 100 nodes is shown in Figure 2. The number of data packets received at the sink increases linearly with time for RFSN.

Recall that REEF has different phases. In REEF, data is received only during the steady state. The packets received in REEF are fused data packets and each packet contains readings of all nodes in the network. Hence the number of data packets received at the sink in REEF are less than in RFSN. Note that the sink doesn't get any information from the nodes during the setup phases.

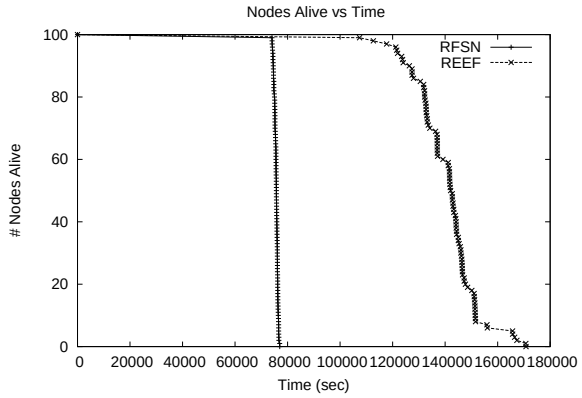


Fig. 4. Number of Nodes Alive vs. Time

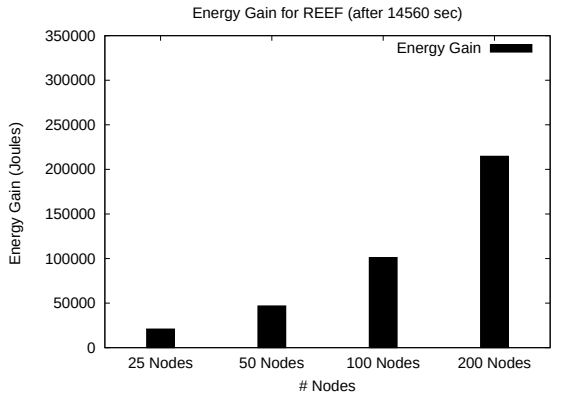
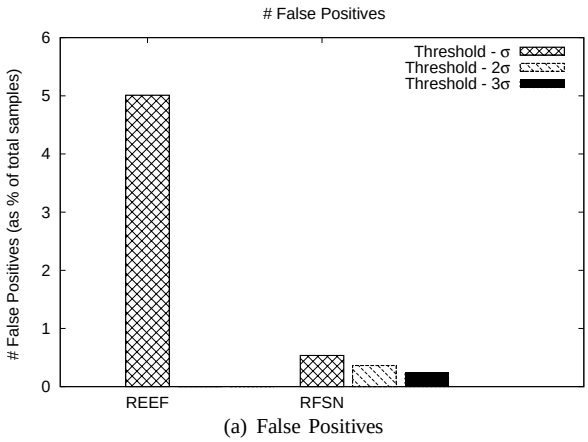
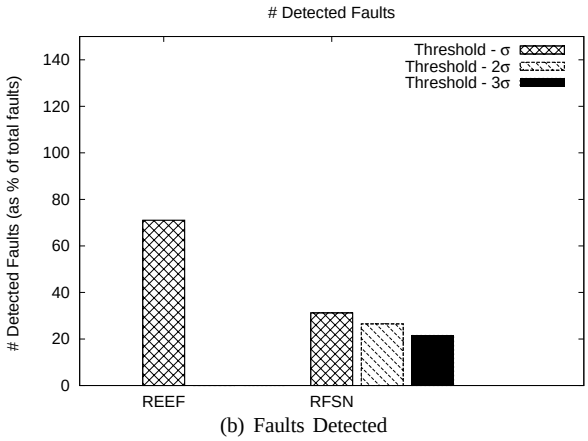


Fig. 5. Energy Gain for REEF vs. Network Size



(a) False Positives



(b) Faults Detected

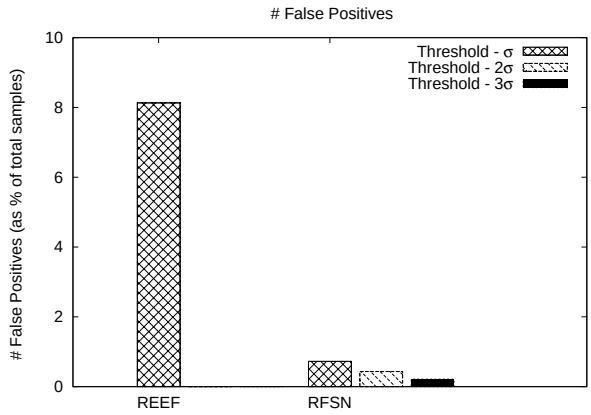
Fig. 6. Performance of REEF and RFSN for short fault for network size 50

3) Energy Spent vs. Data Received by the Sink : Figure 3 depicts the energy spent per data packet received for RFSN and REEF after 14560 seconds which is equal to the time taken to finish three rounds of REEF. As evident from the figure, the number of data packets reaching the sink is very high in RFSN compared to REEF. This is because REEF uses fused data packets and data is transmitted during the steady phase only. Energy spent in the network for the sink to receive fused data packets in REEF is less than energy spent to receive data packets in RFSN. Fused data packets of REEF contain more information than a single data packet in RFSN. As a result, the energy spent in the network for sink to gather a unit of information is less in REEF.

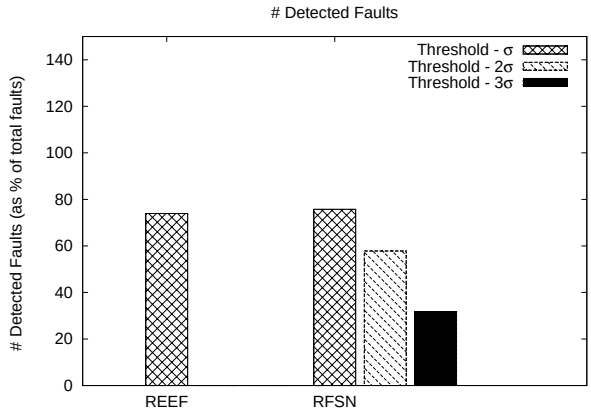
4) Network Lifetime : The network life time is defined as the time for which all the nodes in the network are alive. As time progresses, the energy of nodes get depleted and the nodes die. Figure 4 shows the number of alive nodes as time advances for a network of 100 nodes. In RFSN, the first node dies at 74060.75 seconds and all the nodes are dead by 77022.0 seconds. In REEF, all nodes are alive till 107420.25. Nodes started dying after that and all the nodes are dead by 170802.25 seconds. The network life time is extended in REEF because most of the nodes can go to deep sleep during the steady state.

5) Energy Gain vs. Network Size: Energy spent in the network is less than energy spent by a framework like RFSN which employs passive listening. It was also observed that the Energy Gain by running REEF over running RFSN increases as the network size increases. It is because the energy spent in RFSN increases linearly to the number of nodes. But in REEF, most of the nodes are neither the cluster head nor the gateway and can use deep sleep in steady state. So the energy spent in REEF increase slower than linearly with number of nodes.

6) Fault Detection: Performance comparison of REEF and RFSN in detecting shot faults, stuck-at faults and variance degradation faults are given in Figure 6, Figure 7 and Figure 8. The threshold value used for REEF (used in algorithm 1) is always twice the standard deviation of measurement error of the sensor onboard. For RFSN, the threshold value (used as in equation 1)was $n\sigma$ where $sigma$ is the standard deviation of LOF values and simulations were done with n taking the values 1,2 and 3. Faults detected by RFSN for each of the threshold

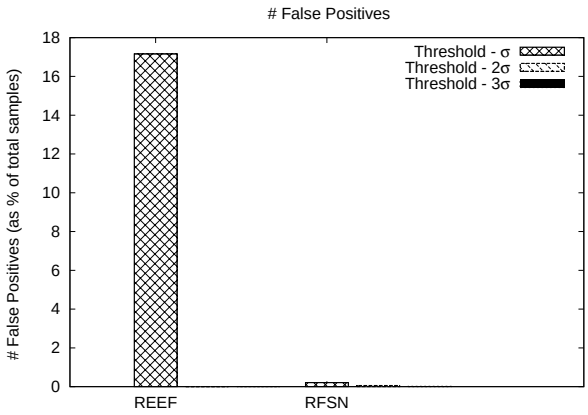


(a) False Positives

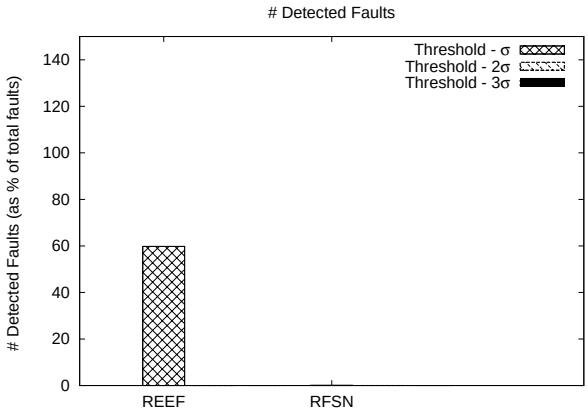


(b) Faults Detected

Fig. 7. Performance of REEF and RFSN for stuck-at fault for network size 50



(a) False Positives



(b) Faults Detected

Fig. 8. Performance of REEF and RFSN for variance fault for network size 50

value was found out. These results are for the network which contains 50 nodes. Results for network with size 25,100 and 200 nodes were found to be similar. For short fault, the number of false positives reported by REEF is more compared to RFSN. But the number of false positives is less than 10 percentage of the total samples. The number of faults correctly detected by REEF is much higher than that detected by RFSN. For stuck-at fault, the number of false positives reported by REEF is more (though it is less than 10 percentage of total samples) and the number of detected faults is less but comparable to that detected by RFSN. In case of variance degradation fault, more than half of the total samples were faulty and RFSN was able to detect less than one percentage of the faults. REEF detected more than 50 percentage of faults, but the number of false positives reported was also high.

V. Conclusion

The total energy spent in the network while using REEF was found to be less than that used by RFSN. Consequently, network life time also got extended when REEF was used. In addition, we observed that the energy gain while using REEF increased with number of nodes in the network. The performance of REEF was comparable with that of RFSN in terms of detecting Short faults and Stuck-At faults. REEF performed better than RFSN in detecting variance degradation faults.

This work focused on sensor networks used for monitoring natural phenomena. It can be extended to sensor network used for surveillance with the help of few key modifications. In this work, clustering was done based on residual energy of the node. To apply this to a network used for surveillance or event detection, the clustering algorithm should take the coverage of the nodes in to consideration so that the area is fully covered even when a large number of nodes are sleeping.

Another possible extension is to improve the outlier detection algorithm to perform better in detecting variance degradation fault.

REFERENCES

[1] C. Chong and S. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247–1256, 2003.

[2] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pp. 88–97, 2002.

[3] L.Balzano, "Addressing fault and calibration in sensor networks," Master's thesis, Dept. of Computer Science, UCLA, 2007.

[4] A. Sharma, L. Golubchik, and R. Govindan, "On the Prevalence of Sensor Faults in Real-World Deployments," *4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pp. 213–222, 2007.

[5] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay *et al.*, "A macroscope in the redwoods," *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pp. 51–63, 2005.

[6] N. Ramanathan, L. Balzano, M. Burt, D. Estrin, T. Harmon, C. Harvey, J. Jay, E. Kohler, S. Rothenberg, and M. Srivastava, "Rapid deployment with confidence: Calibration and fault detection in environmental sensor networks," *Center for Embedded Networked Sensing, UCLA and Department of Civil and Environmental Engineering, MIT, Tech. Rep. CENS Tech Report*, vol. 62, 2006.

[7] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable Coordination in Sensor Networks," *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 263–270, 1999.

[8] V. Mhatre and C. Rosenberg, "Design guidelines for wireless sensor networks: Communication, Clustering and Aggregation," *Ad Hoc Networks*, vol. 2, no. 1, pp. 45–63, 2004.

[9] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient Communication Protocol for Wireless Microsensor networks," *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, p. 10, 2000.

[10] O.Younis and S.Fahmy, "HEED-a hybrid, energy efficient, distributed clustering approach for ad-hoc sensor networks," *IEEE Transactions on Mobile Computing, Vol 3, Issue 4, Oct*, 2004.

[11] ——, "An experimental study of routing and data aggregation in sensor networks," *Proceedings of LOCAN, IEEE MASS, November*, 2005.

[12] S. Basagni, "Distributed clustering for ad hoc networks," *Parallel Architectures, Algorithms, and Networks, 1999.(I-SPAN'99) Proceedings. Fourth InternationalSymposium on*, pp. 310–315, 1999.

[13] S.Mukhopadyay, D.Panigrahi, and S.Dey, "Model based error correction for wireless sensor networks," *IEEE Wireless Communications and Networking Conference, Atlanta, March*, 2004.

[14] Q.Zhang, T.Yu, and P.Ning, "A framework for identifying compromised nodes in sensor networks," *Proceedings of 2nd IEEE Communcation society / SecureComm2006*, 2006.

[15] J.Chen, S.Kher, and A.Somani, "Distributed fault detection of wireless sensor networks," *Proceedings of the 2006 workshop on Dependability issues in wireless ad hoc networks and sensor networks*, 2006.

[16] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, no. 3, pp. 1–37, 2008.

[17] Y. Zhuang and L. Chen, "In-network outlier cleaning for data collection in sensor networks," *CleanDB Workshop*, pp. 41–48, 2006.

[18] P. Rabinovich and R. Simon, "Secure Aggregation in Sensor Networks Using Neighborhood Watch," *Communications, 2007. ICC'07. IEEE International Conference on*, pp. 1484–1491, 2007.

[19] D. Wagner, "Resilient aggregation in sensor networks," *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pp. 78–87, 2004.