ON SUPPORTING DISTRIBUTED COLLABORATION IN SENSOR NETWORKS

Guiling Wang, Wensheng Zhang, Guohong Cao, and Tom La Porta
Department of Computer Science & Engineering
The Pennsylvania State University, University Park, PA 16802

ABSTRACT

In sensor networks, nodes may malfunction due to the hostile environment. Therefore, dealing with node failure is a very important research issue. In this paper, we study distributed cooperative failure detection techniques. In the proposed techniques, the nodes around a suspected node collaborate with each other to reach an agreement on whether the suspect is faulty or malicious. We first formalize the problem as how to construct a dominating tree to cover all the neighbors of the suspect and give the lower bound of the message complexity. Two tree-based propagation collection protocols are proposed to construct dominating trees and collect information via the tree structure. Instead of using the traditional flooding technique, we propose a coverage-based heuristic to improve the system performance. Theoretical analysis and simulation results show that the heuristic can help achieve a higher tree coverage with lower message complexity, lower delay and lower energy consumption.

1. INTRODUCTION

Recent advances in micro-electro-mechanical systems technology, wireless communications and digital electronics have enabled the development of low-cost, low-power and multi-functional sensor nodes. These tiny nodes, which consist of sensing, data processing, and communicating components, further leverage the concept of sensor networks [2],in which a large number of sensor nodes collaborate to monitor certain environment. Sensor networks represent a significant improvement over traditional sensors. They are broadly deployed in many military and civil applications, such as target tracking, surveillance and security management. The massive deployment of sensor networks and their potential applications in a variety of fields have made sensor networks a hot research topic. The relatively large number of nodes, the limited bandwidth available for inter-sensor communication, and the limited battery power on each of the sensor nodes make the design and implementation of sensor networks very complicated and intriguing. In the past years, most researchers have focused their work on energy-efficient MAC protocol [16], data-centric routing and data aggregation [3], [1], scalable hierarchical routing [7], location management and geography informed routing [4], [15]. However, there are still many other research issues to be addressed. Compared with the expensive and rechargeable laptops in ad hoc networks, the sensor nodes are more likely to have problems or die out, and the sensor networks are difficult to manage because of their deployment environment. Thus, how to deal with failures is a very important research issue.

In sensor networks, each sensor node may have two tasks. One is to collect sensing data; the other is to route the data of other nodes to the data center. According to these two tasks, there are two kinds of failures in sensor networks. (1) Node failure: the sensor nodes do not work properly or the sensor nodes maliciously send out wrong data. For example, the sensors may not report the occurrence of an event when monitoring the forest fire, or report wrong data when monitoring the temperature of some field. Note that this kind of node failure can even be a Byzantine failure. (2) Routing misbehavior: the sensors may maliciously drop the packets they should forward.

Many researchers [10], [5], [8] addressed node failures with the approach of failure masking. The idea is to use redundancy to tolerate some level of failures. Although the solution can tolerate failures, the existence of failed sensors degrade the system performance. Incorrect data reported by faulty (or malicious) sensors reduce the accuracy and reliability of the sensing data [8] even though some fault tolerance techniques are used. Further, routing misbehavior cannot be addressed just by fault tolerance. Thus, failure detection, which identifies the faulty nodes, is rather necessary. Shen [14] proposed a centralized approach to detect and identify faulty nodes. In this approach, sensor nodes are asked to send data to a central node, which is responsible for checking the data and identify faulty nodes. This approach has large communication overhead due to propagating queries and transmitting responses. The central node may be overloaded by large communication and computation workload. Furthermore, this approach results in a large failure detection latency, and may interfere with the normal system operations. Shen also proposed solutions to reduce the response implosion by sacrificing some accuracy. To deal with routing misbehavior, Marti *et al.* [9] proposed another approach. After the sender sends the packet to the next node, it keeps listening to the media to see if the next

node really forwards the packet. Although this solution can improve the network performance in the presence of misbehaving (faulty) nodes, it has some drawbacks. Asking the sender to keep listening may consume a large amount of power. Also, as suggested in [9], the misbehaving node may not be able to detect some packet drops due to collisions. Furthermore, misbehaving nodes can also generate fake alerts or falsely accuse other nodes, and eventually disable network operation.

To improve the detection accuracy, we propose to let the neighbors of a faulty node coordinate to detect the problem. With the geographical adjacency, the neighbors of one node may have overlapping sensing range with this faulty node and may have similar observation over the detected objects. For example, in monitoring the temperature of a field, nearby area should have similar temperature and a sudden drop or increase of the temperature may be treated as problems. The neighbors of the node are more qualified and may be the best ones to identify the faulty node since they can find out the abnormality of that node, while remote nodes cannot. Furthermore, as the communication neighbors, they can work together to find out whether the suspected node drops their packets maliciously, since the number of packets sent to this node plus the packets it generates should be equal to the number of packets it sends out when the input packets do not flush the buffer. In order for the neighbors to identify the faulty node collaboratively, we focus on designing protocols for the neighbors of the faulty node to communicate with each other. To achieve this goal, we propose two *Tree-based Propagation-Collection (TPC)* protocols to collect the information from all neighbors of the suspect with low delay, low message complexity, and low energy consumption. Theoretical analysis and extensive simulations results verified that the protocols can indeed achieve good performance.

The rest of the paper is organized as follows. Section 2 presents framework of distributed cooperative failure detection. Section 3 describes the a general TPC protocol, and gives the lower bound of the message complexity. Two distributed TPC protocols are presented in Section 4. Section 5 evaluates the performance of these protocols. Section 6 concludes the paper.

2. THE FRAMEWORK OF FAILURE DETECTION

In this section, we describe the framework of distributed cooperative failure dection. To clearly define the framework, we first present the assumptions.

2.1. ASSUMPTIONS

To clearly define the proposed framework of distributed cooperative failure detection, we have following assumptions.

- A promiscuous mode is supported by the wireless interface. When a node N_i is located within a node N_j's transmission range, N_i can overhear N_j's communication.
- During the process of coordinated failure detection, no new failure occurs and all the detectors are in good condition. This assumption can be relaxed if there are a large number of neighbors.
- Each node has a constant transmission power and the transmission range is constant. We assume the two-ray propagation model [13] for communications with a $\frac{1}{d^4}$ roll-off. This model has been shown to be close to the reality in many environments. A node receiving a message can discover the sender's distance by measuring the received power.
- The neighbors of the suspected node can be communicated without going through the suspected node. This assumption is valid when the sensor density is high, and the assumption has been used in many studies [11].

2.2. THE FAILURE DETECTION FRAMEWORK

The proposed framework is only initiated when necessary to save bandwidth and power. It has three major components:

The first component is *Failure detection without coordination*. To detect routing misbehavior, a node promiscuously overhears the transmission activities of its neighbors. If the number of packets that one neighbor fails to forward exceeds certain threshold, it becomes a suspect. To deal with node failures, a node can overhear the data sent by its neighbors, and compare these data with those sensed by itself. With some application-specific knowledge, such as the upper-bound of the difference between the data reported by two neighbors and the range of the absolute value may be estimated, sensor nodes can estimate the working condition of their neighbors.

The second component is *Collecting results from distributed detection*. When a node suspects that one of its neighbors is faulty, it sends out messages to request the opinions on the behavior of this suspected neighbor from other neighbors of the suspect. All neighbors send back their opinions in response to the request. In the paper, the node to initiate the data collection is referred to as the $initiator(N_i)$. The suspected node is referred to as the $suspect(N_s)$. Any other node is referred to as a *co-detector* (N_c).

The third component is *Diagnosis and notification of the diagnosis result*. After collecting the detection results from all co-detectors, the initiator proceeds on the result to diagnose whether the suspect has a fault. The process depends on some application-specific knowledge. For example, in

monitoring the temperature of a field, nearby area should have similar temperature and a sudden drop or increase of the temperature may be treated as problems. If a node has sent many packets to N_s, but no other node claims to have received any packet from N_s, N_s must have dropped some packets. Also, techniques developed in [14] can be used to help find out if N_s is a faulty node. If the suspect is diagnosed as a faulty node, the diagnosis result is sent to the related nodes.

Among these three components, our paper focuses on the second one, and concentrates on designing efficient communication protocols for the initiator to collect data from co-detectors with low message complexity, delay, and energy consumption.

3. THEORETICAL ANALYSIS

In this section, we give out the lower bound of the message complexity for the initiator to collect the data from all co-detectors. This lower bound is calculated in an ideal situation where each node has the knowledge of the network topology.

Intuitively, this collection procedure can have two steps. First, the initiator propagates the request. Next, the co-detectors sends their information back. Since the initiator may not be able to reach all the co-detectors due to limitation of the communication range, the collection process may have multiple rounds. To save bandwidth and energy, a communication structure should be constructed during the propagation period such that during the second step, messages can go through this structure and necessary aggregation can be made. A tree is our natural communication structure to reach all the co-detectors and the initiator is the root.

Theorem 1: If $T(V_t, E_t)$ is a minimum dominating tree rooted at the initiator, the minimum message complexity for the initiator to collect all the information from the co-detectors is $n + n_t$, where n denotes the number of co-detectors plus one and n_t denotes the size of the dominating tree.

Proof. The proof is by contradiction. Suppose it needs only $m < n + n_t$ messages to collect the opinions from n co-detectors. To collect opinions from n nodes, we need n messages for these n nodes to send out their opinions. Also, we need messages to notify these n nodes. Then, $m - n$ messages are required to notify these co-detectors, where $l = m - n < n_t$. This means that the senders of these l messages can reach all the n co-detectors. Then these l senders can compose a dominating tree which is smaller than the minimum dominating tree. A contradiction.

Theorem 1 gives us the lower bound of the message complexity that can be achievable. It can only be achieved when every node knows the overall topology. In reality, such a topology is unknown, and then constructing a minimum dominating set is known to be NP-hard [6].

4. THE TPC PROTOCOLS

In reality, nodes do not have the global knowledge of the network topology. Different from the theoretical analysis of communication through a dominating tree in a known graph, we need to first find such a graph, or at least the subgraph generated by the $suspect$. In case that the initiator does not know the identities of the co-detectors, by one-step query, it can find the co-detectors among its neighbors. Consequently, the new co-detectors continue querying their neighbors to find out more co-detectors. This procedure repeats until all co-detectors are notified of the detection event and they can send back their opinions to the initiator. Following this general procedure, we propose two protocols: the basic protocol and the coverage-based protocol.

4.1. THE BASIC TPC (B-TPC) PROTOCOL

The propagation procedure of the B-TPC protocol is similar to the flooding algorithm. At the first time when a node receives a request from the initiator or other co-detector, it re-broadcasts the request if it is the neighbor of the $suspect$, and sets the sender of the request message as its parent node. This propagating continues until the number of hops reaches a threshold. Then a dominating tree is formed, and each co-detector sends back their opinions through the tree. To reduce the traffic, some aggregation work can be done. After broadcasting the request, none-leaf nodes set a timer to wait for the data from the sub-tree rooted by them. When the timer expires, it aggregates the collected data, composes them into one packet and sends to its parent.

4.2. THE COVERAGE-BASED TPC (C-TPC) PROTOCOL

In B-TPC, each node needs to re-broadcast the request, which may not be necessary. If a node has more neighbors as co-detectors than others, including this node in the dominating tree can reduce the tree size. This is the main idea on which the coverage-based TPC depends. Now, the question is how a node knows the number of co-detectors within its coverage without using the expensive flooding approach. Our solution is to make use of the $suspect$. The **idea** is to let the $suspect$ broadcast a $request$ message, which starts the data collection. At the same time, the initiator uses the promiscuous mode to monitor whether the suspect really broadcasts the request. As for the suspect, it has to cooperate. If it does not, the initiator knows that the suspect is a faulty node. If the $suspect$ is a malicious node, it may behave normally after it knows that it becomes the suspect. However, this is not a problem since the detection is based on the previous behavior of the suspect, which has been

logged by the initiator or the neighbors. Based on these logs, the initiator can find out if the suspect is malicious or not.

As the suspect sends out the broadcast message, all the co-detectors will be able to hear it and find out that they should be involved in the failure detection. The C-TPC protocol works in three phases:

In the first phase (*Co-detector discovery*), the $suspect$ is required to broadcast a message about the detection initiation. After receiving the message, each co-detector broadcasts a message to declare its identity and the data (related to the suspect) to report. Nodes receiving the declaration message knows their coverage, since receiving a message from a co-detector means that the co-detector is within its own coverage. To reduce collisions, each co-detector backoffs a random amount of time before declaring its identity.

The second phase (*Tree formation*), starts when the initiator broadcasts a request message to ask for the coverage of its neighbors. The receivers will reply sooner or later depending on the additional coverage they can provide. The larger the additional coverage, the quicker they reply. At the same time, they monitor other nodes' reply to find out whether nodes with larger additional coverage already reply. If so, they will cancel their reply messages to avoid collisions. The initiator selects one with the largest additional coverage as its $successor$ to perform the same job as itself. The procedure continues with more and more data collected, until there is no one can provide additional coverage. All nodes who have been selected as successors compose the dominating tree.

The third phase (***Data collection***) is described as follows. If a newly nominated $successor$ can not further expand the dominating tree, it sends the collected data to its parent who will do the same job until the initiator receives such a packet. In this procedure, every node promiscuously overhears the transmission of this final report. If a node has additional coverage which is not included in this report, it reports individually through the dominating tree. The initiator broadcasts a final request to its neighbors to see whether some information are neglected. The nodes that can provide additional coverage becomes a node of the dominating tree. All the nodes belong to the dominating tree are also called *backbone* nodes. The formal description of the protocol is shown below.

Notations:

- d, M, l: the estimate of the maximum additional coverage, the maximum coverage, and possible neglected coverage in the final report.
- $S_c(N_k)$: node N_k's coverage, which is the set of neighbors of both N_k and N_s.

The algorithm at the initiator:

(A) Upon suspecting node N_s:
Ask N_s to broadcast a message to start the detection process.

(B) Upon overhearing N_s's announcement:
Set $Timer$ to be M slots and $State$ to be $SendRequest$.

(C) Upon receiving the final report:
$S_d = S_d \bigcup finalreport$;
Broadcast $request(flag, S_d)$;
Set $Timer$ to be l slots and $State$ to be $WaitTerminate$.

(D) Upon receiving $report(S_c(N_k))$ from N_k:
if $S_c(N_k)$ is larger than that of previous successor candidate, set N_k to be current successor candidate

(E) Upon timeout:
if $State$ = $WaitTerminate$, then terminate the procedure;
if $State$ = $SendRequest$, then
Broadcast $request(S_c(N_i))$;
Set $State$ to be $SelectSuccessor$ and $timer = 2$;
if $State$ = $SelectSuccessor$, then
if the candidate exists, send $Successor(d/2)$ to it;
Terminate if no successor candidate when d slots limit is reached.

The algorithm at the co-detectors

(A) Upon receiving the announcement from N_k:
Broadcast its own identity and data after a random number of slots

(B) Upon receiving identity declaration from Node N_k:
$S_d = S_d \bigcup S_d(N_k)$.
$S_c = S_c \bigcup N_k$

(C) Upon receiving $request(d, S_c(N_k))$ from N_k:
Set $timer$ to be $max(d- \mid S_c - S_c(N_k) \mid, 0)$ number of slots
Set $state$ to $WaitReport$

(D) Upon overhearing $report(S_c(N_k))$ from N_k:
If $S_c(N_k)$ is larger than my additional coverage, cancel the $Timer$

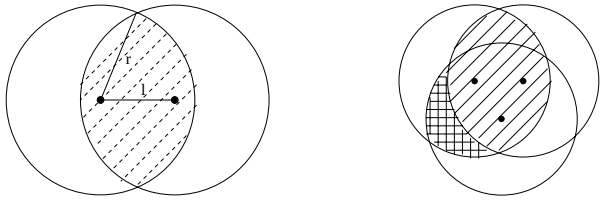
(E) Upon receiving $Successor(d)$ from N_k
$S_d = S_d \bigcup S_d(N_k)$
Broadcast $request(d)$
Set $State$ to be $SelectSuccessor$
Set $timer$ to be 2 slot, after timeout, and this $timer$ will continuously be set one slot until d slots limit is reached.

(F) Upon receiving $report(S_c(N_k))$ from N_k:
Same action as the $Initiator$

(G) Upon overhearing final report or the request with flag:
If I have additional coverage, send it to my parent.

(K) Upon timeout:
if $State$ = $WaitReport$ send $report$ to my parent
if $State$ = $SelectSuccessor$
if Current Successor Candidate exists, send it $Successor(d/2)$.
if no successor candidate when d slots have been waited, send final report to N_i through the constructed dominating tree.

There are several parameters in C-TPC whose value needs to be considered. M is used only by N_i. After overhearing the announcement from N_s, N_i needs to wait some time to broadcast its request so that its neighbors have finished broadcasting their identity. M is the estimate of how long this procedure will last. A large M may result in longer latency and a small value may result in an incomplete coverage and an inaccurate selection of the successor. Suppose it needs one slot to send out a message, and only one message can be transmitted in one neighborhood. After M slots, the initiator can assume that all its neighbors know their coverage. Figure 1 (a) shows how to choose M. The intersection area of these two circles can be calculated as f(l) = $4 * \int_{l/2}^{r} \sqrt{r^2 - x^2}\, dx$, where $l = 0 \cdots r$. The average value of the intersection area is $\int_0^r 2\pi x f(x)/\pi * r^2\, dx \simeq 0.59\pi r^2$. The minimum of the intersection area is $0.39\pi r^2$ when $l = r$. In our simulation, N_i chooses M to be 0.59*the number of its neighbors.



(a) M (b) d

Figure 1. Choosing parameters for the C-TPC protocol

Parameter d is used to estimate the maximum additional coverage of its children for the $backbone$ nodes. For N_i, d can be (1-0.39)*number of neighbors. For the rest of the $backbone$ nodes, d can be half of that of their parent. Dividing d by two in each step is a conservative guess of the additional coverage. The grid shadow part in Figure 1 (b) shows the additional coverage of the second step propagation. Based on some simulation results [12], the average value of this additional coverage is $0.22 * \pi * r^2$. We use $\lceil (1 - 0.39)/2 * n \rceil$ as our estimate.

The minimum message complexity of C-TPC protocol is $n + 4 * n_t - 2$, which can be achieved when each time the node with the largest coverage replies first and all the other nodes overhear it and stop its own transmission. One message is used to instruct the $suspect$. One message is used by initiator to broadcast the final request. $n - 1$ messages is used the co-detectors to broadcast their identities. n_t messages are used to broadcast request, $n_t - 1$ messages are used to compete to be the backbone nodes, $n_t - 1$ messages are used to denominate the new $backbone$ nodes, and another $n_t - 1$ messages are used for the nodes in the dominating tree to send the final report back to the N_i. Certainly, a smaller n_t (the size of the dominating tree) has less message complexity.

5. PERFORMANCE EVALUATIONS

5.1. SIMULATION MODEL

We use ns2 (version 2.1b8a) as our simulator. Our simulation takes place in a 500 by 500 meter flat space. We use 802.11 as the MAC layer protocol. The physical layer works as the 914MHz Lucent WaveLAN DSSS radio interface. The propagation model is Two ray ground and the transmission range is 250 meters.

The goal of the proposed protocols is to collecting information from all co-detectors with low latency, low energy consumption, and low message overhead. Correspondingly, we evaluate our work from these aspects. The coverage of our algorithm has been explained when presenting the protocols. Our simulation results also verify that all co-detector have been covered. Thus, we do not show any figure about this. As for the latency, we measure the delay from the $initiator$ starting the detection to the termination of the algorithm. Normally, the cost of an application includes the bandwidth occupied and the energy consumed. To evaluate the bandwidth usage, we use message complexity as the metric, which is the total message used during the detection procedure divided by the number of neighbors of the $suspect$.

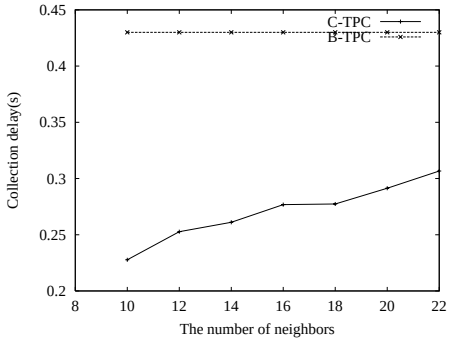


Figure 2. The delay

5.2. SIMULATION RESULTS

5.2.1. THE DELAY

Figure 2 shows the delay of three protocols. Apparently, L-TPC has the shortest delay while B-TPC has the highest one. To achieve full coverage, B-TPC chooses $maxhop = 8$, which is redundant in most of the cases. Thus, the delay of B-TPC is much higher than the other two and this delay does not change in this figure. From this figure, we can see that letting the $suspect$ notify the co-detectors can reduce the delay of failure detection. This explains why the C-TPC approach has much lower delay than the B-TPC approach.

5.2.2. THE MESSAGE COMPLEXITY

Figure 3 shows the message complexity of both algorithms. As shown in the analysis, the lowest message complexity achievable by C-TPC is $n+4*n_t-1$ and the highest

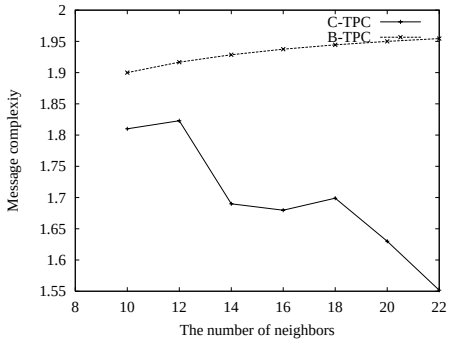


Figure 3. The message complexity

message complexity is $2n + 3 * n_t - 1$. Its message complexity depends both on the setting of the parameters and the topology. When several nodes have the same additional coverage, they all send reply to the $backbone$ nodes. When M slots are not sufficient to finish the identity broadcasting, the coverage information is not accurate, and consequently n_t can be larger. When d is not large enough, several nodes may reply simultaneously. From Figure 3, we can see that the message complexity of C-TPC is not very stable with a higher node density. B-TPC always has the highest complexity, which is $2n - 1$. We also need to note that with a higher node density, the message complexity of C-TPC is close to the lower bound of the ideal situation.

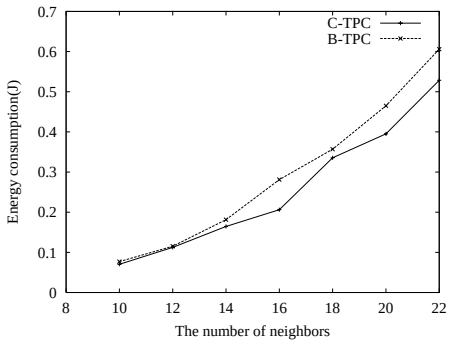


Figure 4. The energy consumption

5.2.3. THE ENERGY CONSUMPTION

Figure 4 shows the energy consumption of both algorithms. This figure is easy to understand after observing the previous two figures. When the message complexity is high, more energy are consumed as long as the timer is properly set to reduce the chances of collisions. From the figure, we can see that the difference of energy consumption among these two is not that significant compared to the difference of the message complexity. The reason is that both approaches use broadcasts, and the energy consumption during idle time is similar to that during sending or receiving.

6. CONCLUSIONS

This paper defined a framework for distributed cooperative failure detection in sensor networks. Within this framework, we study protocols for nodes around a suspected node to communicate with each other to reach an agreement. A family of tree-based propagation-collection (TPC) protocols have been proposed to address the problem. In the B-TPC protocol, requests are flooded to the neighbors of the suspect to construct a dominating tree that includes all the neighbors of the suspect. Facilitated by the tree, detection information is collected and processed, and an agreement can be reached. However, this protocol has high latency, message complexity and energy consumption. To address these drawbacks, a heuristic based approach was proposed. The C-TPC protocol uses the coverage information as a heuristic to select nodes that have large coverage to construct a smaller dominating tree. Simulation results show that the C-TPC protocol has lower delay, less message complexity and energy consumption compared to the B-TPC protocol.

References

[1] C. Intanagonwiwat , R. Govindan and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication," *MobiCOM '00*, August 2000.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, March 2002.

[3] B.Krishnamachari, D. Estrin and S. Wicker, "Modelling Data-Centric Routing in Wireless Sensor Networks," *IEEE INFOCOM'02*, June 2002.

[4] N. Bulusu, D. Estrin, L. Girod and J. Heidemann, "Scalable Coordination for Wireless Sensor Networks: Self-Configuring Localization Systems," *ISCTA 2001*, July 2001.

[5] P. Chew and K. Marzullo, "Masking Failures of Multidimensional Sensors," *Proc. of the 10th Symposium on Reliable Distributed Systems (SRDS'91)*, pp. 32–41, 1991.

[6] B. Clark, C. Colbourn and D. Johnson, "Unit Disk graphs," *Discrete Mathematics*, pp. 165–177, 1990.

[7] W. R. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor network," *Proc. of the Hawaii International Conference on System Sciences*, January 2000.

[8] D. Jayasimha, "Fault Tolerance in a Multisensor Environment," *Proc. of the 13th Symposium on Reliable Distributed Systems (SRDS'94)*, October 1994.

[9] S. Marti, T. Giuli, K. Lai and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," *ACM MobiCom*, Aug. 2000.

[10] K. Marzullo, "Tolerating Failures of Continuous-valued Sensors," *ACM Transactions on Computer Systems*, November 1990.

[11] D. Nicules and B. Nath, "Ad-hoc Positioning System using AoA," *IEEE INFOCOM*, 2003.

[12] S. Ni,Y. Tseng, Y. Chen and J. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," *ACM Mobicom*, 1999.

[13] T. S. Rappaport, "Wireless communications: principles and practice," 1996.

[14] C. Shen, "Diagnosis of Sensor Networks," *IEEE International Conference on Communications (ICC'01)*, June 2001.

[15] Y. Xu, J. Heidemann and D. Estrin, "Geography Informed Energy Conservation for Ad Hoc Routing," *ACM MOBICOM'01*, July 2001.

[16] W. Ye, J. Heidemann and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," *IEEE INFOCOM'02*, June 2002.