Using SensorRanks for In-Network Detection of Faulty Readings in Wireless Sensor Networks

Xiang-Yan Xiao, Wen-Chih Peng and Chih-Chieh Hung
Hsinchu, Taiwan, ROC
{xyxiao, wcpeng, hungcc}@cs.nctu.edu.tw

Wang-Chien Lee
The Pennsylvania State University
State College, PA 16801, USA
wlee@cse.psu.edu

ABSTRACT

In this paper, the problem of determining faulty readings in a wireless sensor network without compromising detection of important events is studied. By exploring *correlations* between readings of sensors, a correlation network is built based on similarity between readings of two sensors. By exploring Markov Chain in the network, a mechanism for rating sensors in terms of the correlation, called *SensorRank*, is developed. In light of SensorRank, an efficient in-network voting algorithm, called *TrustVoting*, is proposed to determine faulty sensor readings. Performance studies are conducted via simulation. Experimental results show that the proposed algorithm outperforms *majority voting* and *distance weighted voting*, two state-of-the-art approaches for in-network faulty reading detection.

Categories and Subject Descriptors: H.3.4 [Systems and Software]: Distributed Systems
General Terms: Algorithms, Design, Reliability
Keywords: faulty readings, wireless sensor networks

1. INTRODUCTION

Sensors are prone to failure in harsh and unreliable environments. Faulty sensors are likely to report arbitrary readings which do not reflect the true state of environmental phenomenon or events under monitoring. Meanwhile, sensors may sometimes report noisy readings resulted from interferences [3]. Both arbitrary and noisy readings are viewed as *faulty readings* in this paper. The presence of faulty readings may cause inaccurate query results and hinder their usefulness. Thus, it is critical to identify and filter out faulty readings so as to improve the query accuracy.

In this paper, we target at the problem of determining faulty readings in sensor networks. Obviously, a naive approach to this problem is to collect all readings to a sink, where statistical analysis is performed to determine what readings are outliers. However, this centralized approach may not be practical due to limited energy budget in sensor nodes. If readings are sent to the sink all the time, sensor nodes may soon exhaust their energy. Nevertheless, simply filtering out unusual readings at individual sensor nodes may compromise monitoring accuracy of some important events. The goal of this study is to design an energy efficient in-network algorithm for determining faulty readings without compromising detection of important events.

The fact that data readings of nearby sensors are similar can be captured by *spatial correlation* [6]. Thus, an idea for determining faulty readings is to exploit this spatial correlation. In other words, if a sensor obtains an unusual reading, the sensor could inquire its nearby sensors (referred to as the *witness set*) by sending the suspected reading to them in order to determine whether the reading is faulty or not. Based on the classical *majority voting*, each sensor (e.g., sensor s_i) in the witness set makes a judgment by comparing its own reading with the unusual reading sent by the suspected sensor (e.g., sensor s_j). If the difference between these two readings exceeds a predefined threshold, s_i considers the reading sent by s_j as faulty and gives a negative vote to s_j. Otherwise, s_i claims that s_j is normal and returns a positive vote to s_j. After collecting votes from the nearby sensors, s_j then can conclude whether the reading is faulty or not. If the number of negative votes is smaller than that of positive votes, the unusual reading reported by s_j is identified as a faulty reading. Otherwise, it is viewed as an observed event. However, this simple majority voting approach does not work well when the number of faulty sensors increases. To address the problem, two *weighted voting* methods has been proposed in the literature [5, 9]. Motivated by an assumption that the closer sensors have more resemble readings, the weighted voting algorithms give more weights to closer neighbors in voting (i.e., the weights are assigned inverse to the distances from a sensor node to its neighbors).

In this paper, however, we argue that the distance between two sensors does not fully represent the correlation between readings of those two sensors. Furthermore, if the nearest sensor is faulty, the voting result may be seriously contaminated by this faulty sensor. We refer to this problem as a *domination problem* in the paper. Figure 1 illustrates a sensor network where the neighboring sensor nodes are linked. Each link is labeled by a weight (determined based on heuristics adopted by different voting methods) that will be used in voting. Assume that the weights of sensors s_2, s_3 and s_4 are 0.3, 0.4 and 0.9, respectively, and sensor s_4 is a faulty sensor. Obviously, the reading of sensor s_1 is identified as a faulty reading when the weighted voting method is

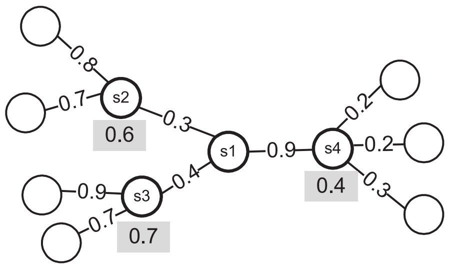

Figure 1: An illustrative topology of a wireless sensor network.

performed (i.e., 0.3*1+0.4*1+0.9*(-1)=-0.2) where positive and negative votes are represented by 1 and -1, respectively).

As shown above, the distance-based weighted voting method has two primary deficiencies: 1) while the distance between sensor nodes may be a factor in generating similar readings of nearby sensor nodes, it does not precisely capture what we really care about as the *correlation* between sensor readings; 2) while it is a good idea to inquire opinions of neighbors, the *trustworthiness* of neighbors is not considered.

Based on the above observation, in this paper, we propose an innovative in-network voting scheme for determining faulty readings by takeing into account the correlation of readings between sensor nodes and the trustworthiness of a node. To obtain the pair-wise correlations of sensor readings, we construct a logical *correlation network* on top of the sensor network. The correlation network can be depicted by a set of vertices and edges, where each vertex represents a sensor node and a edge between two sensor nodes denotes their correlation (i.e., the similarity between their readings). If two nearby sensor nodes do not have any similarity in their readings, these two sensor nodes do not have an edge connected. Therefore, only sensor nodes that are connected by correlation edges are participated in voting. The weighted voting method actually uses the correlation (modeled as a function of distance) between sensor nodes as weights. However, using the correlation alone may not correctly identify faulty readings due to the domination problem discussed above. Thus, in the proposed algorithm, each sensor node is associated with a trustworthiness value (called *SensorRank*) that will be used in voting. SensorRank of a sensor node implicitly represents the number of 'references' (i.e., similar sensor nodes nearby) it has to support its opinions. A sensor node will obtain a high SensorRank if this sensor has many references. The number under each sensor node in Figure 1 is its SensorRank. In the figure, s_4 has a small SensorRank because the readings in s_4 are not very similar to that of its neighbors. By using SensorRank, our voting scheme takes the trustworthiness of each sensor into account. A vote with small SensorRank has only a small impact on the final voting result. For example, in Figure 1, when s_1 inquires opinions from its neighbors (i.e., s_2, s_3 and s_4), the vote from s_4 has a small impact due to its lower SensorRank.

Our design consists of two parts: 1) an algorithm that calculates SensorRank for each sensor node; and 2) an algorithm that use SensorRank to determine faulty readings. Specifically, we first obtain correlations among sensor readings and then model the sensor network as a Markov chain to determine SensorRank. In light of SensorRank, the *TrustVoting* algorithm we developed will be invoked as needed in operation to effectively determine faulty readings. A preliminary performance evaluation is conducted via simulation. Experimental result shows that the proposed TrustVoting algorithm is able to effectively identify faulty readings and outperforms *majority voting* and *distance weighted voting*, two state-of-the-art voting schemes for in-network faulty reading detection for sensor networks.

A significant amount of research effort has been elaborated upon issues of identifying faulty sensor readings [2, 5, 6, 9]. In [6], the authors explored spatial correlation among sensors and proposed a distributed Bayesian algorithm for detecting faulty sensors. By assuming that faulty measurements are either much larger or much smaller than normal measurements, the authors in [2] use a statistical method to detect outlier measurements. Some variations of the weighted voting technique for detecting faulty sensors are proposed in [5] and [9]. In [5], the past performances of sensors are considered to enhance the classical majority voting, and the coverage of sensing range is considered in [9] for its weighted voting. To the best of our knowledge, prior works neither fully formulate the similarity of sensors nor provide the concept of SensorRank, let alone devising filtering algorithm based on SensorRank. These features distinguish this work from others.

The rest of this paper is organized as follows. The notion of correlation network is presented in Section 2. The SensorRank and the TrustVoting algorithms are described in Section 3 and Section 4, respectively. A preliminary performance evaluation is conducted in Section 5. Finally, the conclusion and future work are discussed in Section 6.

2. CORRELATION NETWORK

As mentioned earlier, prior works only take the distance between sensor nodes into consideration when modeling the correlation of sensor readings. However, it is also possible that the readings of two geographically close sensor nodes to have dramatically different readings. Thus, it's critical to truly capture the correlation of sensor readings rather than their distance.

Definition 1. Reading Vector: Assume that the overall readings of a sensor s_i consists of a series of readings in a sliding window Δt. The readings of s_i can be expressed as $b_i(t) = \{x_i(t-\Delta t+1), x_i(t-\Delta t+2), \ldots, x_i(t)\}$, where $x_i(t)$ is the reading sensed by s_i at the time t.

Clearly, the readings of a sensor within a sliding window is represented as a *reading vector*. Therefore, we can define the similarity of two sensor nodes in terms of their reading vectors. Since a faulty reading may be very different from other normal readings from the perspectives of trend and magnitudes, we employ the Extended Jaccard similarity [7] as our similarity function. The Extended Jaccard similarity function for calculating the similarity of two sensors s_i and s_j is denoted as $corr_{i,j}$ and defined as follows:

$$\begin{aligned} corr_{i,j} &= \frac{b_i(t)\cdot b_j(t)}{||b_i(t)||_2^2 + ||b_j(t)||_2^2 - b_i(t)\cdot b_j(t)}, \\ \text{where } ||b_i(t)||_2^2 &= |x_i(t-\Delta t+1)|^2 + \cdots + |x_i(t)|^2 . \end{aligned}$$

When the readings of two sensors have neither the similar trend nor the similar difference, the value of $corr_{i,j}$ is close to 0. On the other hand, the value will be set to 1 when the reading vectors of two sensors are exactly the same.

Assume that the communication range of a sensor node is denoted as R and the geographical distance of two sensor nodes is represented as $dist(s_i, s_j)$. In light of the correlations among sensor nodes in the network, a correlation network is defined as follows:

Definition 2. Correlation network: The correlation network is modeled as a graph $G = (V, E)$, where V represents the sensor nodes in the deployment region and $E = \{(s_i, s_j)|s_i, s_j \in V,\ dist(s_i, s_j) < R \text{ and } corr_{i,j} > 0\}$. The weight of an edge (s_i, s_j) is assigned to be $corr_{i,j}$.

Once the correlation network of sensors is constructed (and maintained), one can easily deduce the correlations among sensor nodes. Based on the correlation network, we shall further develop an algorithm to compute SensorRank for each sensor node, in terms of the correlation with its neighbors, in the network.

3. SENSORRANK

SensorRank is to represent the trustworthiness of sensor nodes. By our design, two requirements need to be met in deriving SensorRank for each sensor.

Requirement 1: If a sensor has a large number of neighbors with correlated readings, the opinion of this sensor is trustworthy and thus its vote deserves more weight.

Requirement 2: A sensor node with a lot of trustworthy neighbors is also trustworthy.

These two requirements ensure that 1) a sensor node which has a large number of similar neighbors to have a high rank; and 2) a sensor node which has a large number of 'good references' to have a high rank. Given a correlation network $G = (V,\ E)$ derived previously, we determine SensorRank for each sensor to meet the above two requirements.

We model the correlation network as a Markov chain M, where each sensor s_i is viewed as the state i, and the *transition probability* from state i (i.e., sensor s_i) to state j (i.e., sensor s_i) is denoted as $p_{i,j}$ and formulated as $p_{i,j} = \frac{corr_{i,j}}{\sum_{k \in nei(i)} corr_{i,k}}$. For example, in Figure 2, $p_{2,3} = \frac{0.1}{0.4+0.1+0.7} = 0.083$. Based on the above setting, we can formulate SensorRank of s_i, denoted as rank$_i$, as follows:

$$rank_i = \sum_{s_j \in nei(i)} rank_j \cdot p_{j,i}$$

where $nei(i)$ is the witness set of node i.

The computation of SensorRank can be viewed as a random walk over the correlation network. Several iterations is required to perform random walks until a steady state is achieved (i.e., SensorRanks become stable). Specifically, $rank_i^{(k)}$ is the value of SensorRank at the k-th iterations. At the beginning, the initial $rank_i^{(0)}$ is set to 1. Note that $rank_i^{(0)}$ can be set to any constant c, and the results will be c times the value generated when the initial SensorRank is set to 1. In the first round, each sensor node s_i updates its SensorRank as $rank_i^{(1)}$ using the initial SensorRanks of its neighbors. Now each sensor node has considered the first level neighbors to calculate its SensorRank. In the second round, each sensor node can indirectly obtain some information from the second level neighbors through its first level neighbors since its first level neighbors have explored their first level neighbors as well. Therefore, after the kth round, sensor node s_i has explored the kth level neighbors and updated SensorRank as $rank_i^{(k)}$.

Algorithm 1 *SensorRank*

Input: a sensor s_i, and a threshold δ.
Output: $rank_i$ for s_i.

```
1: $rank_i^{(0)} = 1$
2: for $k = 1$ to $\delta$ do
3:     for all $s_j \in nei(s_i)$ do
4:         $p_{i,j} = \frac{corr_{i,j}}{\sum_{s_k \in nei(i)} corr_{i,k}}$
5:         send $rank_i^{(k-1)} \cdot p_{i,j}$ to $s_j$
6:     receive all $rank_j^{(k-1)} \cdot p_{j,i}$ from every $s_j \in nei(i)$
7:     $rank_i^{(k)} = \sum_{s_j \in nei(i)} rank_j^{(k-1)} \cdot p_{j,i}$
```

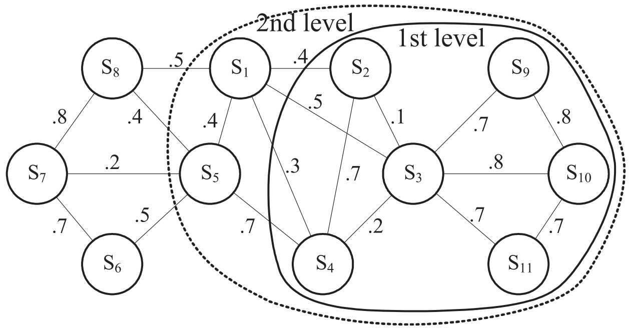

Figure 2: An example of Sensor Rank.

Consider an example in Figure 2. In the first round, s_3 has some similarity information from its first level neighbors $\{s_2, s_4, s_9, s_{10}, s_{11}\}$. Similarly, both s_2 and s_4 could exchange some information with their neighbors. In the second round, s_3 can obtain similarity information from the second level neighbors $\{s_1, s_5\}$ since its first level neighbors s_2 and s_4 have explored s_1 and s_5 during the first round. If k is larger, SensorRanks will be more accurate since every sensor can explore more neighbors. In sensor networks, the computation cost will be larger when the number of iterations is larger. Therefore, we can limit k to a preset bound δ.

Given a correlation network in Figure 2, we now demonstrate how to calculate SensorRank. Initially, sensor s_i sets its sensorRank $rank_i^{(0)}$ to 1. For sensor s_i, s_i calculates the trust relations $p_{i,j}$ to the corresponding neighbor s_j and sends $rank_i^{(0)} \cdot p_{i,j}$ to s_j. For example, s_3 sends $rank_3^{(0)} \cdot p_{3,1} = 1 \cdot \frac{0.5}{3.0} = 0.167$ to s_1, 0.033 to s_2, $\frac{0.2}{3} = 0.067$ to s_4, and etc. At the same time, s_3 receives SensorRanks from its neighbors. For example, s_3 receives $rank_2^{(0)} \cdot p_{2,3} = 1 \cdot \frac{0.1}{0.4+0.1+0.7} = 0.083$ from s_2. Upon receiving all the proportion of SensorRank from the neighbors, s_3 can update its SensorRank to $rank_3^{(1)}$.

$$\begin{aligned} rank_3^{(1)} &= \sum_{i \in \{1,2,4,9,10,11\}} rank_i^{(0)} \cdot p_{j,i} \\ &= 1 \cdot p_{1,3} + 1 \cdot p_{2,3} + 1 \cdot p_{4,3} + 1 \cdot p_{9,3} \\ &\quad +1 \cdot p_{10,3} + 1 \cdot p_{11,3} \\ &= \frac{0.5}{2.1} + \frac{0.1}{1.2} + \frac{0.2}{1.9} + \frac{0.7}{1.5} + \frac{0.8}{2.3} + \frac{0.7}{1.4} \\ &= 1.74 \end{aligned}$$

After the first round, $\left\{rank_i^{(1)}|i = 1,2,3,4\right\}$ ={1.13, 0.59, 1.11, 1.33}. In the second round, sensors calculate the values

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
$k=0$	1	1	1	1	1	1	1	1	1	1	1
$k=1$	1.13	0.59	1.74	1.11	1.33	0.64	1.14	0.89	0.58	1.3	0.54
$k=2$	1.17	0.68	1.43	1.05	1.24	0.77	0.91	1.05	0.86	1.04	0.8

Table 1: SensorRank values for sensors in Figure 2.

of SensorRank with the updated values of SensorRank in the first round. For example, s_1 now sends $rank_1^{(1)} \cdot p_{1,3} = 1.13 \cdot \frac{0.5}{2.1} = 0.269$ to s_3. Similarly, when s_3 receives all the values from its neighbors, s_3 can update its SensorRank to $rank_3^{(2)}$. Assume that $\delta = 2$, s_1 will stop updating its SensorRank, and $\left\{rank_i^{(2)} | i = 1, 2, 3, 4\right\}$ ={1.17, 0.68, 1.43, 1.05}. As expected, s_3 has the highest SensorRank 1.43, since s_3 has many similar neighbors. Since s_1 has fewer similar neighbors than s_3, s_1 has smaller SensorRank than s_3. The values of SensorRank after the third iteration are listed in Table 1. From Table 1, s_3 has the largest SensorRank since more nearby sensors have similar reading behaviors with s_3. This meets the requirements we set for design of SensorRank as mentioned earlier.

4. TRUSTVOTING ALGORITHM

Here we describe our design of the *TrustVoting* algorithm, which consists of two phases: a) self-diagnosis; and b) neighbors diagnosis phase. In the self-diagnosis phase, each sensor verifies whether the current reading of a sensor is unusual or not. Once the reading of a sensor goes through the self-diagnosis phase, this sensor can directly report the reading. Otherwise, the sensor node consults with its neighbors to further validate whether the current reading is faulty or not. If a reading is determined as faulty, it will be filtered out. The sensor nodes generating faulty readings will not participate in voting since these sensors are likely to contaminate the voting result. Note that TrustVoting is an in-network algorithm which is executed in a distributed manner. The execution order of algorithm TrustVoting has an impact on faulty reading detection. We will discuss this issue later.

Algorithm 2 $TrustVoting$

```
Input: a sensor $s_i$, SensorRank $rank_i$ and time interval $t$
Output: justify whether the reading is faulty or not (i.e.,
    $faulty = true$ or not)
 1: set $faulty = false$
 2: broadcast $rank_i$ to the neighbors
 3: receive $\{rank_j | s_j \in nei(i)\}$ from the neighbors
    /* set timer by the priority sorted by SensorRank */
 4: sort SensorRank values received
 5: $x = rank_i$'s order in the sorted SensorRank values
 6: $n$ = neighbors of sensor $s_i$
 7: $timer = x * (\frac{t}{n+1})$ /*$t$ is the time interval given */
 8: while $time == timer$ do
 9:    $faulty$ = Procedure Self-Diagnosis
10:    if $faulty == true$ then
11:       $faulty$ = Procedure Neighbor-Diagnosis
12:       return $faulty$
```

4.1 Self-diagnosis Phase

When a set of sensor nodes is queried, each sensor in the queried set performs a self-diagnosis procedure to verify whether its current reading vector is faulty or not. Once the reading vector of a sensor node is determined as normal, the sensor node does not need to enter the neighbor-diagnosis phase. To execute a self-diagnosis, each sensor s_i only maintains two reading vectors: i) the current reading vector at the current time t (denoted as $b_i(t)$); and ii) the last correct reading vector at a previous time t_p (expressed by $b_i(t_p)$). $b_i(t_p)$ records a series of readings occurred in the previous time and is used for checking whether the current reading behavior is faulty or not. If these two reading vectors are not similar, $b_i(t)$ is viewed as an unusual reading vector. Once a sensor node is detected an unusual reading vector, this sensor node will enter the neighbor-diagnosis phase to further decide whether the unusual reading behavior is faulty or not. Note that when $b_i(t)$ is identified as a normal vector through the neighbor-diagnosis, $b_i(t_p)$ is updated so as to reflect the current monitoring state.

4.2 Neighbor-diagnosis Phase

If a sensor node s_i sends $b_i(t)$ to a neighbor s_j, s_j will compare $b_i(t)$ with its own current reading vector $b_j(t)$ and then give its vote with respect to $b_i(t)$. From the votes from neighbors, s_i has to determine whether $b_i(t)$ is faulty or not. Notice that some votes are from sensors with high SensorRank. A sensor node with high SensorRank has more similar neighbors to consult with and thus is more trustworthy. Therefore, the votes from the neighbors with high SensorRank are more authoritative, whereas the votes from the neighbors with low SensorRank should cast less weights.

When sensor s_i sends $b_i(t)$ to all its neighbors for the neighbor-diagnosis, each neighbor should return its vote after determining whether $b_i(t)$ is faulty or not. If a neighbor s_j considers $b_i(t)$ is not faulty by comparing the similarity of the two reading vectors (i.e., $corr_{i,j} \geq \sigma$), s_j will send a positive vote, denoted $vote_j(i)$, to s_i. Otherwise, the vote will be negative. In addition, the vote from s_j will be weighted by its SensorRank.

$$vote_j(i) = \begin{cases} rank_j, & corr_{i,j} \geq \sigma \\ -rank_j, & \text{otherwise.} \end{cases}$$

After collecting all the votes from the neighbors, s_i has two classes of votes: one is positive class ($b_i(t)$ is normal) and the other is negative class ($b_i(t)$ is faulty). If the weight of the former is larger than the weight of the later, the most neighbors will view $b_i(t)$ as normal. Note that the weight of a vote represents how authoritative a vote is. It is possible that a neighbor s_j of s_i with a large SensorRank has a small correlation with s_i. In this case, these two sensor nodes may not provide good judgments for each other. Therefore, each vote (i.e., $vote_j(i)$) has to be multiplied by the corresponding correlation, $corr_{i,j}$. Thus, we use the following formula to determine whether the reading is faulty or not.

$$dec_i = \sum_{s_j \in nei(i)} corr_{i,j} \cdot vote_j(i)$$

Procedure *Neighbor-Diagnosis*

```
Input: a sensor $s_i$, its current reading behavior $b_i(t)$, and a threshold $\sigma$.
Output: the variable $faulty$.
 1: set $dec_i = 0$
 2: broadcast $b_i(t)$ to the neighbors
 3: for all $s_j \in nei(i)$ do
 4:     if $sim(b_i(t), b_j(t)) \geq \sigma$ then
 5:         $vote_j(i) = rank_j$
 6:     else
 7:         $vote_j(i) = -rank_j$
 8:         $dec_i = dec_i + tr_{ij} \cdot vote_j(i)$
 9: if $dec_i \geq 0$ then
10:     return $false$
11: else
12:     return $true$
```

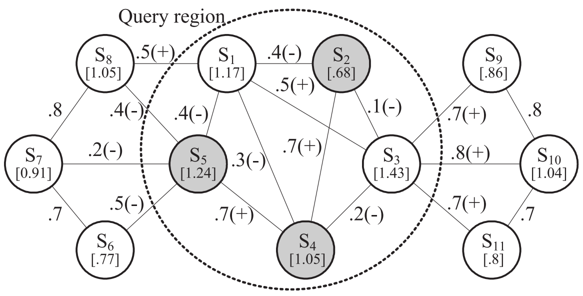

Figure 3: An example query for TrustVoting.

If the weight of the positive votes are more than the weight of the negative votes, dec_i will be positive which means that s_i's reading is normal and the current reading can be reported. Otherwise, dec_i is negative, implying that the current reading of s_i is faulty. For example in Figure 3, a region of sensors is queried (s_1, s_2, s_3, s_4 and s_5) and four faulty sensors (gray nodes) exist. SensorRanks of sensors are shown in square brackets in nodes and the correlation between sensors are shown on edges. To facilitate presentation of this example, the plus sign (minus sign) shows that two sensor nodes have similar (dissimilar) current readings, and they are going to give the positive (negative) votes to each other when executing the neighbors' diagnosis. Consider sensor node s_5 as an example, where s_5 will receive the votes from its neighbors (i.e., s_1, s_4, s_6, s_7 and s_8). It can be obtained that $dec_5 = (-1.17) \cdot 0.4 + 1.05 \cdot 0.7 + (-0.77) \cdot 0.5 + (-0.91) \cdot 0.2 + (-1.05) \cdot 0.4 = -0.72$. Therefore, the reading reported by sensor node s_5 is a faulty reading.

4.3 Execution Order of TrustVoting

Since the TrustVoting algorithm is a distributed algorithm, sensors being queried will perform the self-diagnosis and neighbor-diagnosis procedures individually. Different execution orders have produce different results for faulty detection. For example, consider two execution orders $\{s_1, s_2, s_3, s_4, s_5\}$ and $\{s_5, s_1, s_2, s_3, s_4\}$ in Figure 3. Assume that all the queried sensor nodes have to perform the neighbors' diagnosis. In the order of $\{s_1, s_2, s_3, s_4, s_5\}$, when s_1 executes TrustVoting, s_2, s_4 and s_5 give negative votes, while s_3 and s_8 claim positive votes. As such, dec_1 will be 0.16 and s_1 will be identified as normal. For s_2, since

Order	Faulty	Not faulty
s_1, s_2, s_3, s_4, s_5	s_5	s_1, s_2, s_3, s_4
s_5, s_1, s_2, s_3, s_4	s_4, s_5	s_1, s_2, s_3

Table 2: Faulty detection under different orders.

$dec_2 = (-1.17) \cdot 0.4 + (-1.43) \cdot 0.1 + 1.05 \cdot 0.7 = 0.124$, s_2 is identified as normal. Following the same operations, we find that s_4 is also identified as normal. However, in the order of $\{s_1, s_2, s_3, s_4, s_5\}$, the result is different. When s_5 executes TrustVoting, s_5 is identified as faulty obviously because almost all neighbors give s_5 negative votes. Therefore, s_5 do not vote for other sensors. Without the vote from s_5, s_4 is regarded as faulty since $(-1.17) \cdot 0.3 + 0.68 \cdot 0.7 + (-1.43) \cdot 0.2 = -0.161$. Table 2 shows the results under two different execution orders. From Table2, not all faulty readings reported by faulty sensors (i.e., s_2, s_4 and s_5) are detected and difference executions orders have an impact on the faulty reading detection.

As such, how to determine an appropriate order to perform self-diagnosis and neighbor-diagnosis in algorithm TrustVoting will have an impact on the final result. Since algorithm TrustVoting is executed in a distributed manner, we could use a timer to control the execution order of procedures self-diagnosis and neighbor-diagnosis. Those sensors having smaller values in their timers will perform first. By exploring SensorRank, we could allow those sensor nodes with higher SensorRank to perform self-diagnosis and neighbor-diagnosis as soon as possible. As pointed out early, sensor nodes with a high SensorRank are likely to have many similar neighbors, thereby these sensors could be correctly identified whether readings are faulty or not. Once sensors reporting faulty readings are detected, these sensors do not get involved in voting in other sensor nodes. Therefore, the domination problem can be alleviated since those faulty sensors with higher weights could be determined as early as possible.

Clearly, we could determine the order of executing procedures of self-diagnosis and neighbor-diagnosis according to SensorRank. However, some highly ranked sensor nodes may get their ranks from their highly ranked neighbors while having few neighbors. Therefore, the number of neighbors should also be taken into consideration. In algorithm TrustVoting, timers are set for each sensor in accordance to both of the SensorRank and the number of neighbors. Specifically, assume that a time interval will be given in algorithm TrustVoting. In algorithm TrustVoting, each sensor should first broadcast SensorRank to neighbors. Once receiving SensorRank values from its neighbors, each sensor should sort SensorRank values in a decreasing order. Then, each sensor should determine the order of its SensorRank in such sorted list. Furthermore, a sensor will have information related to the number of neighbors from SensorRank values received. Therefore, we could set timer to be $x \cdot \frac{t}{(n+1)}$, where x is the order of this sensor in a sorted list, n is the number of neighbors and t is the time interval given. With a smaller value of timer, procedures of self-diagnosis and neighbor-diagnosis will executed first.

Consider an illustrative example in 3. The timer value for sensor s_3 should be $1 \cdot \frac{t}{7}$ since sensor node s_3 has 6 neighbors and its SensorRank is the highest among SensorRank values collected (i.e., 6 neighbors and sensor s_3). Following the same operation, we could have the timer values $\frac{2\cdot t}{6}$, $\frac{3\cdot t}{3}$, $\frac{3\cdot t}{3}$, and $\frac{2\cdot t}{6}$ for s_1, s_2, s_4 and s_5, respectively. Assume that each sensor does not pass self-diagnosis and have to execute the neighbor-diagnosis procedure. According to the timers derived, s_3 will perform first and the reading of s_3 is identified as a normal reading by neighbor-diagnosis Then, both s_1 and s_5 will execute neighbor-diagnosis at the same time. The reading reported by s_1 (respectively, s_5) will be determined as a normal reading (respectively, faulty). The executions of s_2 and s_4 are then performed. In particular, since s_5 is viewed as faulty, s_5 could not participate in voting process of s_4. As a result, through the execution order derived, we could accurately detect faulty readings reported by s_2, s_4 and s_5.

5. PERFORMANCE EVALUATION

5.1 Simulation Model

We simulate a synthetic environment, where sensors are deployed in a 500 by 500 to monitor temperatures. The temperature reading range is $[-25, 275]$. Moreover, events with unusual readings are randomly generated in the monitored field. The model of generating events are the same as in [5, 6]. The faulty sensor rate (abbreviated as faulty rate) is the ratio of the number of faulty sensors and the total number of sensors deployed. Each sensor will report noisy readings according to the parameter *noise_prob*. A faulty sensor always report faulty readings and thus *noise_prob* is set to 1 for faulty sensors. On the other hand, a normal sensor is still likely to report noise or faulty readings. Thus, for normal sensors, we set the *noise_prob* to 0.1. A noise reading (referred to as a faulty reading) is randomly biased from the normal reading generated and the amount of bias is within the range of $[-50, 50]$. A query is submitted to wireless sensor networks with its query region as a rectangle and query region size varied from 80 by 80 up to 160 by 160. To evaluate the simulation result, two performance metrics are employed: *faulty detection rate* and *false positive rate*. Specifically, a query is issued to a query region B to obtain the current readings sensed by the sensors, where the set of these current readings is denoted as X_B. Assume that Y_B is a set of faulty readings in X_B. After executing TrustVoting algorithm, we can filter out a set of faulty readings denoted as Y_B', and obtain a subset of current readings $X_B' \subseteq X_B$ without faulty readings. The faulty detection rate is defined as $\frac{|Y_B \cap Y_B'|}{|Y_B|}$ and the false positive rate is defined as $\frac{|(Y_B \cup Y_B') - (Y_B \cap Y_B')|}{|X_B|}$. In other words, the faulty detection rate is the percentage of faulty readings correctly identified, and the false positive rate is the percentage of faulty readings (respectively, normal readings) that are identified as normal (respectively, faulty) readings. We implement the classical majority voting (denoted as MajorVoting) and the distance weighted voting (denoted as WeightVoting) for comparison.

5.2 Simulation Result

5.2.1 *Performance of TrustVoting*

First, we evaluate the performance of these three algorithms. The length of reading vectors for a sensor node is set to 5 and the similarity threshold is set to 0.5. For TrustVoting, the number of iterations for calculating SensorRank is set to 3. Figure 4 shows the faulty detection rates of these three algorithms with various faulty rates. It can be seen that TrustVoting can detect almost 90% faulty readings while MajorVoting and WeightVoting can only identify 40% faulty readings. However, since faulty readings in our faulty model are biased from normal readings, it is hard to identified faulty readings for MajorVoting and WeightVoting. By exploring SensorRank, TrustVoting outperforms other two voting algorithms. Figure 5 shows the false positive rate of the three algorithms. As the faulty rate increases, false positive rates of three algorithms tend to increase due to a larger number of faulty sensors (i.e., it is hard to correctly detect faulty sensors when the number is large).

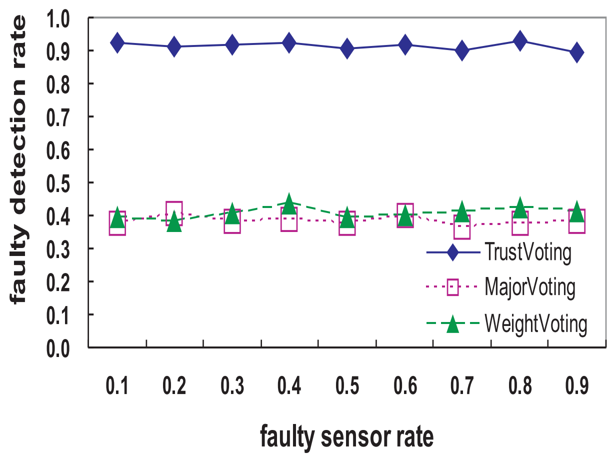

Figure 4: Faulty detection rates of the three algorithms.

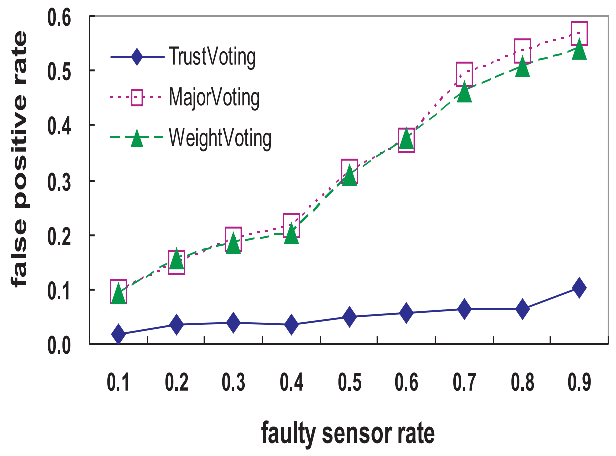

Figure 5: False positive rates of the three algorithms.

5.2.2 *Impact of Iterations on SensorRank*

As mentioned before, SensorRank is calculated iteratively. We now examine the impact of the number of iterations (i.e., the parameter δ) to TrustVoting. Specifically, faulty rates are set to 0.4, 0.5 and 0.6. The length of reading behaviors is set to 5 and the similarity threshold is set to 0.5. The experimental results are shown in Figure 6 and Figure 7. It can be seen in Figure 6, when δ increases, the faulty detection rate will increase. This is due to that with a larger number of iterations, SensorRank is able to have more neighboring information. Therefore, TrustVoting is able to precisely identify faulty readings. Furthermore, with the number of iterations increases, the false positive rate declines. However, increasing the number of iterations for SensorRank will incur message transmissions among sensors. In addition, from Figure 6 and Figure 7, it can be seen that after 3 iterations, the improvements in the faulty detection rate and the false positive rate are not very significant. Therefore, in the following experiments, we set to number of iterations for SensorRank to be 3. Clearly, the number of iteration for SensorRank will be dependent upon the sensing data and can be empirically determined.

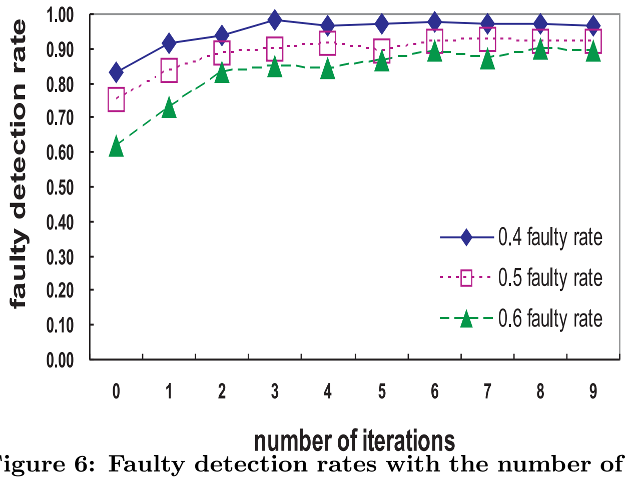

Figure 6: Faulty detection rates with the number of iterations of SensorRank varied.

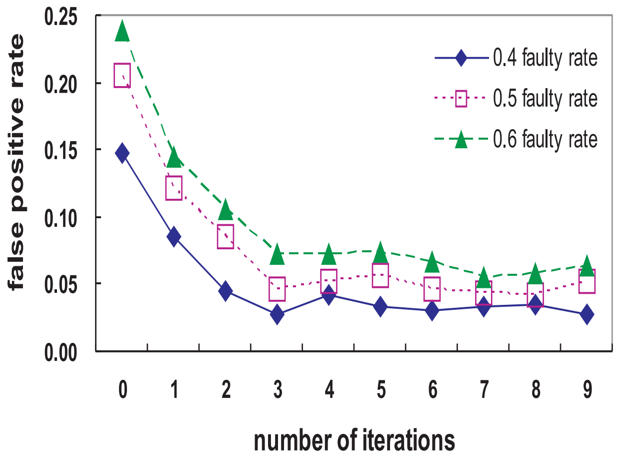

Figure 7: Faulty positive rates with the number of iterations of SensorRank varied.

5.2.3 *Impact of Reading Behavior Length*

As mentioned earlier, reading of sensors is viewed as a series of sensing readings within a sliding window Δt. Then, we conduct experiments to show the impact of Δt. Without loss of generality, the number of iterations for SensorRank is set to 3 and the similarity threshold is set to 0.5. The experimental results are shown in Figure 8 and Figure 9.

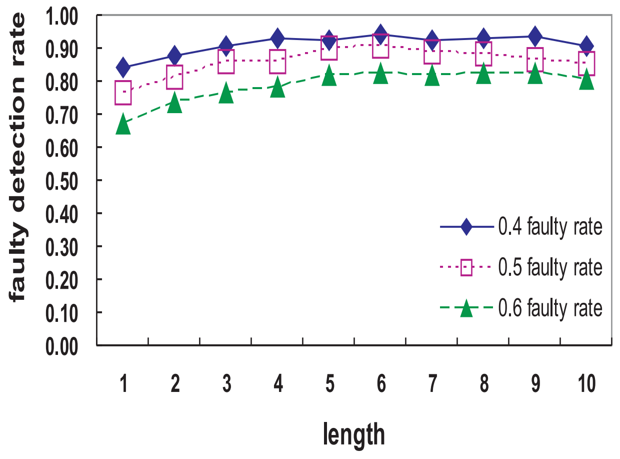

Figure 8: Faulty detection rates with Δt varied.

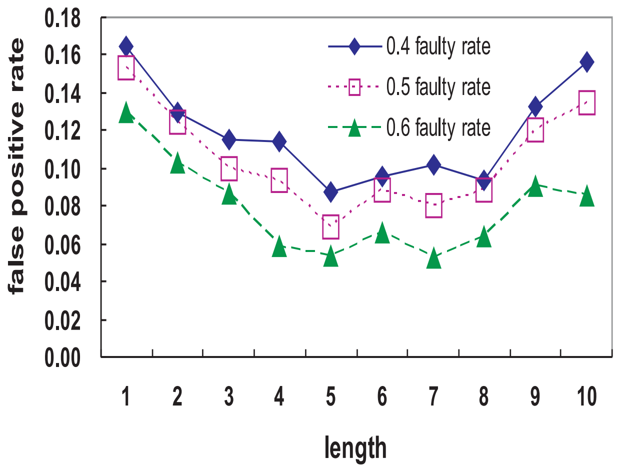

Figure 9: False positive rates with Δt varied.

As can be seen in Figure 8 and Figure 9, the selection of Δt should judiciously be determined. In Figure 8, the faulty detection rate tends to increase with the length of reading behavior. However, the improvement is not significant with larger values of the Δt. On the other hand, in Figure 9, the false positive rate decreases when the length of the reading vectors increases. However, when Δt is larger than 5, the false positive rate is increased. Intuitively, when Δt is small, there are not enough readings for modeling the similarity among sensors, whereas with a larger value of Δt, the reading vectors of sensors may have more noisy readings. Therefore, the setting of Δt is also application dependent and should judiciously selected from the experiments.

5.2.4 *Impact of Neighbors*

Since TrustVoting is a voting scheme, the number of neighbors will have impact on the effectiveness of TrustVoting.

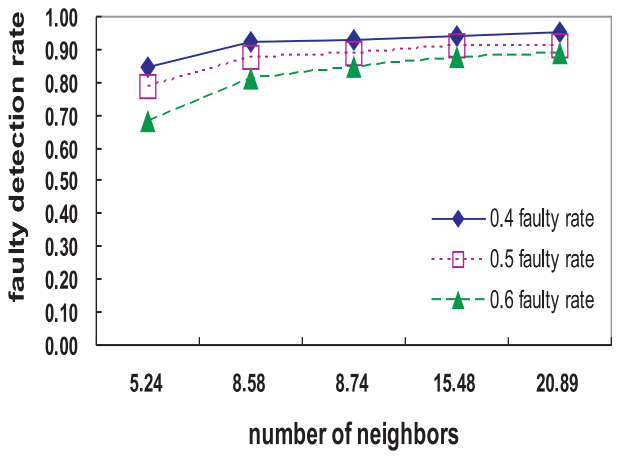

Figure 10: Faulty detection rates with the number of neighbors varied.

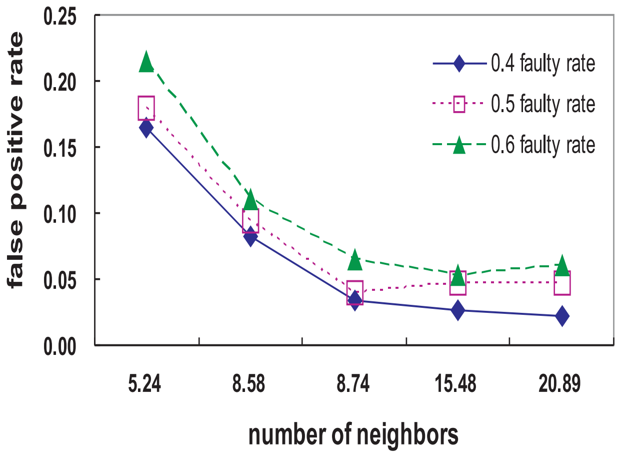

Figure 11: False positive rates with the number of neighbors varied.

Hence, experiments of TrustVoting with the number of neighbors varied are conducted. We set the number of iterations for SensorRank to 3, the similarity threshold to 0.5 and the length of reading vectors to 5. Figure 10 and Figure 11 show the performance of TrustVoting. It can be seen that in Figure 10, the faulty detection rate increases with the number of neighbors. Moreover, in Figure 11, the false positive rate decreases as the number of neighbors increases. With larger number of neighbors, both the faulty detection rate and the false positive rate are greatly improved even for the case in which the faulty rate is high (i.e., 0.6 in this experiment). This agrees with our intuition that the more neighbors a sensor has, the better performance a voting scheme is. The above experiments show that when the faulty rate is high, one should deploy a sufficient amount of sensors so as to further improve the effectiveness of TrustVoting.

6. CONCLUSIONS

With the presence of faulty readings, the accuracy of query results in wireless sensor networks may be greatly affected. In this paper, we first formulated the correlation among readings of sensors nodes. Given correlations among sensor nodes, a correlation network is built to facilitate derivation of SensorRank for sensor nodes in the network. In light of SensorRank, an in-network algorithm TrustVoting is developed to determine faulty readings. Performance evaluation shows that by exploiting SensorRank, algorithm TrustVoting is able to efficiently identify faulty readings and outperforms majority voting and distance weighted voting, two state-of-the-art approaches for in-network faulty reading detection.

Acknowledgement

Wen-Chih Peng was supported in part by the National Science Council, Project No. NSC 95-2211-E-009-61-MY3 and NSC 95-2221-E-009-026, Taiwan, Republic of China. Wang-Chien Lee was supported in part by the National Science Foundation under Grant no. IIS-0328881, IIS-0534343 and CNS-0626709.

7. REFERENCES

[1] T. Clouqueur, K. K. Saluja, and P. Ramanathan. Fault tolerance in collaborative sensor networks for target detection. *IEEE Transactions on Computers*, 53(3):320–333, 2004.

[2] Min Ding, Dechang Chen, Kai Xian, and Xiuzhen Cheng. Localized fault-tolerant event boundary detection in sensor networks. In *Proc. of IEEE INFOCOM*, March 2005.

[3] Eiman Elnahrawy and Badri Nath. Online data cleaning in wireless sensor networks. In *Proc. of International Conference on Embedded Networked Sensor Systems(SenSys)*, pages 294–295, 2003.

[4] Farinaz Koushanfar, Miodrag Potkonjak, and Alberto Sangiovanni-Vincentelli. Fault tolerance in wireless ad-hoc sensor networks. In *Proc. of IEEE International Conference on Sensors*, June 2002.

[5] Mark D. Krasniewski, Padma Varadharajan, Bryan Rabeler, Saurabh Bagchi, and Y. Charlie Hu. Tibfit: Trust index based fault tolerance for arbitrary data faults in sensor networks. In *Proc. of International Conference on Dependable Systems and Networks*, pages 672–681, 2005.

[6] B. Krishnamachari and S. Iyengar. Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *IEEE Transactions on Computers*, 53(3):241–250, 2004.

[7] Alexander Strehl, Joydeep Ghosh, and Raymond J. Mooney. Impact of similarity measures on web-page clustering. In *Proc. of AAAI Workshop on AI for Web Search*, pages 58–64, July 2000.

[8] Sharmila Subramaniam, Themis Palpana, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proc. of International Conference on Very Large Data Bases(VLDB)*, pages 187–198, 2006.

[9] Tony Sun, Ling-Jyh Chen, Chih-Chieh Han, and Mario Gerla. Reliable sensor networks for planet exploration. In *Proc. of International Conference on Networking, Sensing and Control*, pages 816–821, 2005.