Reputation-based Framework for High Integrity Sensor Networks

Saurabh Ganeriwal, Laura K. Balzano, Mani B. Srivastava

Sensor network technology promises a vast increase in automatic data collection capabilities through efficient deployment of tiny sensing devices. The technology will allow users to measure phenomena of interest at unprecedented spatial and temporal densities. However, as with almost every data-driven technology, the many benefits come with a significant challenge in data reliability. If wireless sensor networks are really going to provide data for the scientific community, citizen-driven activism, or organizations which test that companies are upholding environmental laws, then an important question arises: How can a user trust the accuracy of information provided by the sensor network? Data integrity is vulnerable to both node and system failures. In data collection systems, faults are indicators that sensor nodes are not providing useful information. In data fusion systems the consequences are more dire; the final outcome is easily affected by corrupted sensor measurements, and the problems are no longer visibly obvious.

In this paper, we investigate a generalized and unified approach for providing information about the data accuracy in sensor networks. Our approach is to allow the sensor nodes to develop a community of trust. We propose a framework where each sensor node maintains reputation metrics which both represent past behavior of other nodes and are used as an inherent aspect in predicting their future behavior. We employ a Bayesian formulation, specifically a beta reputation system, for the algorithm steps of reputation representation, updates, integration and trust evolution. This framework is available as a middleware service on motes and has been ported to two sensor network operating systems, TinyOS and SOS. We evaluate the efficacy of this framework using multiple contexts: (1) a lab-scale testbed of Mica2 motes, (2) Avrora simulations, and (3) real data sets collected from sensor network deployments in James Reserve.

Categories and Subject Descriptors: ... [**...**]: ...

General Terms: ...
Additional Key Words and Phrases: Faults, Outlier Detection, Reputation, Sensor Networks, Security, Trust.

---

1. INTRODUCTION

The ability of a sensor network to perform its task depends not only on its ability to communicate among the nodes of the network, but also on its ability to sense the physical environment and collectively process the sensed data. Fusing data from multiple sensor nodes in order to produce meaningful results for the end-user implies a reliance on the nodes to provide correct information. However, sensor nodes are implemented using inexpensive hardware components which are highly unreliable. Faulty data can thus result from corruptive processes in the hardware. In our

---

...

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2007 ACM 1529-3785/2007/0700-0001 $5.00

deployment experience, faulty data has often confounded results; We describe this motivation in Section 2. Lest the readers think that these barriers may disappear with advancing technology, we note that the driving force in sensor networks is the large quantity of sensor nodes. Less emphasis is put on the quality of each individual sensor node. Moore's law will push towards ever-cheaper systems. If we are able to understand the limits of these unreliable technologies, then sensor networks will always be able to operate at this lowest-cost boundary.

Therefore, mechanisms in sensor networks which ensure the reliable relaying of data between trusted parties should also account for accuracy and trustworthiness of the data itself. However, network reliability *alone* cannot be used to solve this problem because of uncertainties and the lack of control over the physical world and faulty nodes. Providing a reliable sensing channel is much harder than providing reliability over wireless radio channels because the sensor network designer only has control over the sensing receiver[1].

In this paper, we investigate a generalized and unified approach for providing data authentication in sensor networks. We choose to address the issue of trust in the sensor network by developing a community of sensor nodes. In a human social community, trust between two individuals is developed based on their actions over time. When faced with uncertainty, individuals trust and rely on the actions and opinions of others who have behaved well in the past. The intent is to develop a similar Reputation-based Framework for Sensor Networks (RFSN), where each sensor node develops a reputation for each other node by making direct observations about these other nodes in the neighborhood. This reputation is used to help a node evaluate the trustworthiness of other sensor nodes and make decisions within the network. Reputation systems have been widely studied in the context of several diverse domains such as e-commerce [Resnick and Zeckhauser 2000; Resnick et al. 2000], Internet [Blaze et al. 1996; Blaze et al.], ad-hoc wireless networks [Buchegger and Boudec 2002; Michiardi and Molva 2002], peer-to-peer networks [Xiong and Liu 2003], etc.

Although reputation systems are widely used in diverse domains, none of the existing systems can be used in the context of sensor networks. E-commerce systems, such as ebay [Resnick and Zeckhauser 2000], Yahoo auctions [Resnick et al. 2000], and Internet based systems such as Keynote [Blaze et al. 1996; Blaze et al.], maintain reputation metrics at a centralized trusted authority. Additionally, they use a deterministic number for representing reputation. As a result, these systems use several debatable heuristics for the key steps of reputation updates and integration. In fact, much closer to our context are reputation systems such as those designed for ad-hoc networks, Confidant [Buchegger and Boudec 2002] and Core [Michiardi and Molva 2002], and peer-to-peer networks [Xiong and Liu 2003]. These systems are distributed and also maintain a statistical representation of the reputation by borrowing tools from the realms of game theory. A fundamental difference between these proposals and RFSN lies in the underlying model of node compromise. These schemes are specifically geared towards countering selfish misbehavior and thus, the end objective is to enforce the nodes to cooperate with each other. An inherent assumption is that all nodes are rational. On the contrary, faulty sensor nodes can operate in a completely arbitrary manner.

---

[1] There are some exceptions to this, such as RFID and ultrasound or laser ranging.

This observation led us to instead turn to the economics tool decision theory. Decision theory can be best visualized as the study of games against nature, where nature is an opponent that does not seek to gain the best payout, but rather acts randomly. This general definition allows us to capture many possible node failures such as system faults, process noise, etc. As we will show, reputation representation, updates and integration are straightforward when modeling the problem in realms of decision theory. We note that the possibility of modeling this problem in the domain of game theory should not be disregarded. There is a branch of game theory, non-cooperative game theory [Basar and Olsder 1999], which deals with exactly the same scenario of non-rational participating entities. It will be an interesting future challenge to explore, whether the same can be achieved using non-cooperative game theory, and how this approach performs compared to our approach.

As we will show, RFSN integrates tools from statistics and decision theory into a distributed and scalable framework. We employ a Bayesian formulation, specifically a beta reputation system, for the algorithm steps of reputation representation, updates, integration and trust evolution. This output metric of trust can be used by a node in several ways. For example, a data reading reported by a node can be weighted by the trust of the node when aggregating data from several nodes, thus reducing the impact of the faulty readings. Additionally, the evolution of trust over time provides an on-line tool to the end-user to detect compromised or faulty nodes. This can help the end-user to take appropriate countermeasures such as replacing the corrupted node or sensor.

In Section 3, we will analyze the statistical foundations behind each of the algorithm steps. Furthermore, we will show that all these steps can be reduced to a few basic mathematical operations of addition, subtraction, multiplication and division; making it completely feasible to run RFSN on resource constrained devices. RFSN is available as a middleware service on Motes. It has been ported to two sensor network operating systems: TinyOS [Hill and Culler 2001] and SOS [Han et al. 2005]. In this paper, we will focus on the prototype implementation that has been developed for SOS. We will explain the APIs exposed by RFSN and how sensor network applications can utilize this service in Section 4. In Section 5, we will evaluate the efficacy of RFSN over a lab-scale testbed of Mica2 Motes and also in the context of real data sets collected from deployments in James Reserve. Specifically, we will demonstrate how RFSN can be used effectively in identifying faulty nodes as well as in calculating resilient data aggregates.

2. MOTIVATION

The prime objective of developing sensor networks is that information that can be leveraged from numerous sensing devices placed close to the actual physical phenomena is going to be more accurate then that provided by a few expensive state-of-the-art sensing devices. As a result, the past few years have seen a number of prototype sensor network deployments in diverse domains, a few notable ones being James Reserve [jam], Great Duck Island [Szewczyk et al. 2004], Intel Berkeley [int], CENS deployment at Bangladesh [Ramanathan et al.], etc. However, all these deployments have been done in academic, scientific or engineering con-

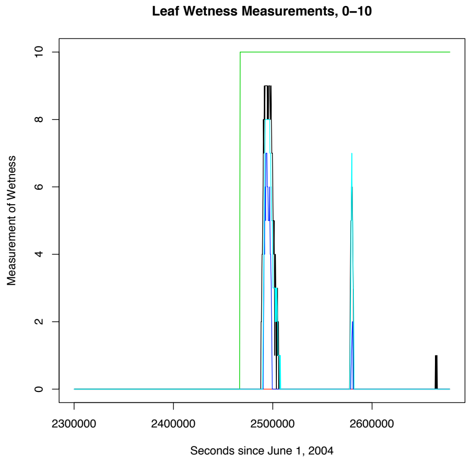


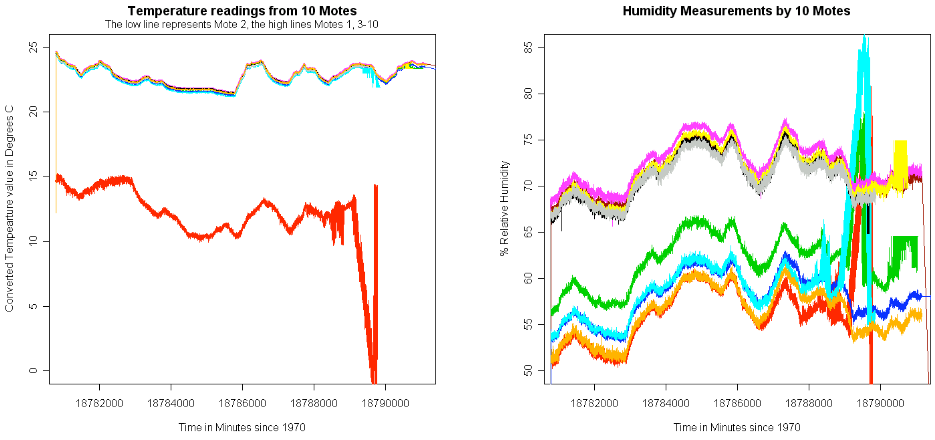
Fig. 1. Leaf Wetness measurements from several weather stations at the James Reserve. A single station got stuck at the highest possible reading.

texts, wherein they operate under controlled environments. As a result, security is a non-issue. Nevertheless, these deployments have still witnessed a large number of misbehaving nodes, arising due to faults in software and hardware. In this section, we list down some of these faults and the arising problems as a result of these faults.

Sticky values are one of the most common faults in sensor network data. The sensor gets stuck at a wrong data value and remains there either permanently or intermittently. The first example of this fault is from a leaf wetness sensor at the James Reserve in California. There are several stations around the reserve measuring leaf wetness on a scale from one to ten. As you can see in Figure 1, one of the sensors got stuck at 10 and remained there for several days. This is a faulty measurement. On the other hand, many of the other sensors measured 0 for extended periods of time, and this is *not* a faulty measurement. Identifying which stuck measurements are faulty and which are not will need more than a simple graphical visualization of the data. We need explicit mechanisms for data measurements to be classified as outliers, depending on expectations given by the application context, measurements taken by other nodes, and other cross-validating information.

A benefit offered by sensor networks is indeed the redundancy in sensors across space. Sensors can thus be used to validate one another. Our second example comes from an experiment to find the variability across temperature and humidity sensors used at the James Reserve. All sensors were closed together in a styrofoam box. In the instance shown in Figure 2, the temperature and the humidity sensors were shorted to one another, causing the temperature measurement to track the wrong phenomenon. In this graph it is clear that the temperature measurement from that sensor device is completely inconsistent with the other temperature measurements.

In the same experiment a more complex situation arises with the behavior of the humidity sensors. During this same calibration experiment, the batteries drained from the motes and behavior would vary at the end of the mote life. Sometimes the



(a) Temperature (b) Humidity

Fig. 2. Temperature and Humidity data from an experiment with ten sensors inside a styrofoam box. One temperature sensor was shorted to a humidity sensor and thus gave readings entirely inconsistent with the other temperature sensors. The humidity sensors experienced inconsistent behavior near the end of the battery life.

radio would die and we would stop receiving data, and these behaviors occured at varying low battery voltages. Other times, the sensor would have a failure response to the low battery before the radio failed. This graph shows some failure examples of the capacitive humidity sensor. This sensor requires a 3.3V excitation voltage. At the point that the sensors were failing, the batteries on the mote were down to 2.3-2.6V. The incorrect behavior can be seen in the right panel of Figure 2. Note that the corrupted value from the humidity sensor was still within a valid range, and hence a simple range check would not identify this fault. Additionally, these measurements are within the range of the correct measurements. We address this in Section 5.

Figure 3 shows temperature measurements from a set of motes in a valley at the James Reserve. The values are all similar, except for those values from one mote which get stuck at a low value and then return to normal. Intermittently stuck values may be indicative of a deeper problem in the sensor, and this sensors' measurements should not be trusted as much as measurements from the others. To accurately establish the relative magnitude of trust between the different sensor nodes, we need mechanisms that take into account the behavior of the node in the past.

The last example of a sticky value is also an example of an out-of-range data fault. Expected ranges are often associated with a phenomenon of interest. The range could be a physical range, such as humidity that can not be outside the range of 0-100 Percent Relative Humidity. Other ranges may be expected range, such as a case where someone deploying the network knows the phenomenon will not exceed that range or the sensor is constructed so as to read the phenomenon of interest within a particular range of values. Ion Selective Electrodes, for example,

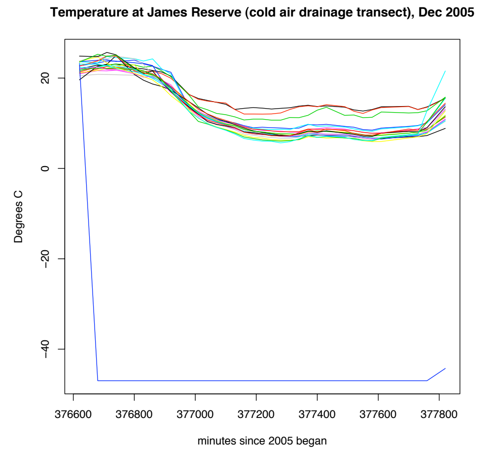


Fig. 3. Temperature measurements from a transect of motes in a valley at the James Reserve. One of the motes' measurements got stuck and then returned to normal.

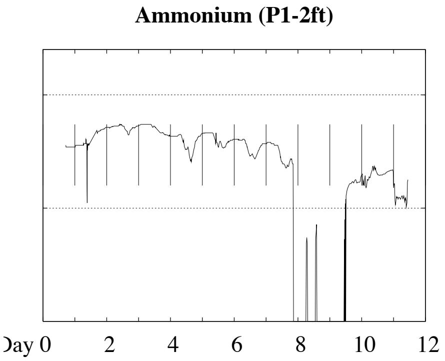


Fig. 4. Concentration of Ammonium measured at 2 ft depth. The vertical lines represent the range of linear sensor response, and the horizontal lines represent the total response range. Many measurements were outside this range.

have a range of voltages over which they have a linear response to a change in phenomenon, a voltage range over which they display a non-linear response, and a range where a voltage measurement does not map to anything at all. Figure 4 shows a dataset collected to measure ammonium concentrations in soil in a rice paddy in Bangladesh. Many measurements were outside the response range of the sensor.

Unlike the James Reserve sensor network deployment, this deployment at Bangladesh corresponds to the scenario wherein a temporary deployment, lasting for a few days, was done to gather as much scientific data as possible. In both the cases, the faults were identified after post processing the collected data from the site of deployment. This was a major loss of deployment effort; although the researchers were present physically at the site, a major bottleneck they faced was the lack of the availability

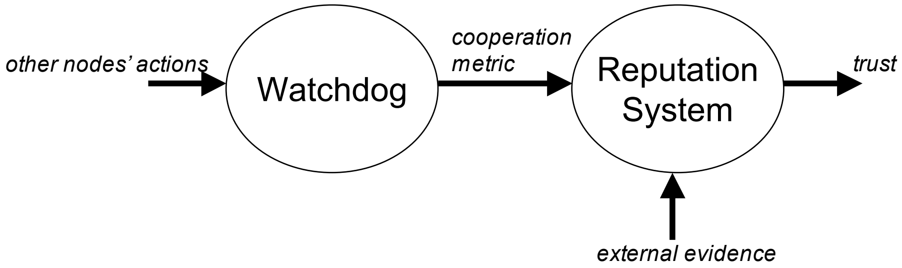


Fig. 5. RFSN Block Diagram

of tools to identify the misbehaving nodes in real time. If that would have been possible, these researchers could have perhaps been able to rectify these failures at the time of deployment (change sensors, node orientation, batteries etc.) and the unnecessary wastage of resources could have been avoided.

The occurrence of so many instances of corrupted data in existing sensor network deployments provides us the motivation to identify misbehaving nodes and inconsistent measurements in real time. In Section 5, we will revisit some of these faults and will try to showcase how these deployments can benefit by using RFSN.

3. REPUTATION BASED FRAMEWORK FOR SENSOR NETWORKS (RFSN)

To address the sensor malfunction problems we have shown, we propose a framework where sensor nodes maintain reputation for other nodes in the network. A sensor node continuously builds these reputation metrics for other nodes by monitoring their behavior and rating them as being cooperative (expected behavior of the nodes in the network) or non-cooperative (unexpected behavior that is most likely the result of a system fault or node compromise). Then the node uses this reputation to evaluate the trustworthiness of other nodes and the data they provide. For example, a node can weigh the data provided by the other nodes with their corresponding reputation metrics while calculating the aggregates or these reputation metrics can be used to identify misbehaving nodes in realtime.

3.1 Overview

Figure 3.1 depicts the two key building blocks of RFSN, Watchdog and Reputation, and their inputs and outputs. The direction of the arrow represents the flow of information.

The block labeled *Watchdog* is responsible for monitoring the actions of other nodes and characterizing these actions as cooperative or non-cooperative. In our work, the actions of interest are sensor readings[2]. Instead of classifying a sensor reading as either cooperative or non-cooperative, we aim to associate a level of confidence or probability (any real number between (0,1)) with it. As can be anticipated, a one-fit all solution does not work for the Watchdog mechanism. Different applications might have different criteria for associating this level of confidence. Therefore, we visualize the Watchdog mechanism to be a library of outlier detection protocols. In Section 3.2, we will discuss one possible protocol in detail, the one which we have used in all our evaluations. As explained in Section 4, the design of the Watchdog mechanism allows easy integration of customized outlier detection protocols for specific application scenarios.

---

[2] In general, the Watchdog can encompass multiple layers such as routing, time synchronization, localization, etc.

The *Reputation* block is responsible for maintaining the reputation of a node. This duty encompasses many tasks. This block manages *reputation representation*, *updates* reputation based upon the new observations made by the Watchdog, *integrates* the reputation information based on other available information, *ages* the reputation, and creates an output metric of *trust*. In the next two subsections, we will explain each of these two blocks in detail.

Before discussing the pieces of the framework, we briefly address the issue of scalability. RFSN runs on each sensor node, and functions in a completely distributed and localized manner. Clearly, it will be infeasible for a node to maintain the reputation for all the nodes in the network. However, unlike peer-to-peer networks, a sensor node interacts most often only with its neighbors[3]. Therefore, a node monitors and maintains reputation for only the nodes in its neighborhood. We note that even for those rare scenarios wherein a node needs to interact with a distant node, it can use a chain through trusted nodes to evaluate its trustworthiness. This provides scalability to the complete architecture.

3.2 Watchdog

The objective of the Watchdog mechanism is to detect the presence of invalid data resulting from compromised and faulty nodes. This block is designed as a library of outlier detection protocols. The input to this block is a set of data readings and the output is a rating for each data reading. This rating, represented by p, is a number in the range $(0, 1)$. It can be inferred as the level of confidence that can be associated with a given data reading.

Outlier detection has been a widely researched problem in the data mining community [Barnett and Lewis]. Only recently, a vast variety of these protocols have been studied in the context of calibrating the sensors [Feng et al. 2003; Koushanfar et al. 2004; Ramanathan et al.] or detecting sensor faults [Koushanfar et al. 2002; 2003; Koushanfar and Potkonjak 2005]. Outlier detection protocols can be broadly classified as either model-based or consensus-based. Model-based outlier detection protocols tend to look for outliers or data readings that deviate from a given model. As an example for sensor networks, this model can represent the underlying physical process or the correlation across different sensing modalities. Each data reading is evaluated independently against a model and as a result, these outlier detection schemes do not put any constraint on the number of total readings or on the goodness of the readings for a correct evaluation. However, a priori knowledge of the expected data model is needed.

On the other hand, consensus-based outlier detection protocols have a common principle of looking for consistency among the data readings in the set. The level of confidence assigned to a data reading is proportional to its deviation from this consensus. Different protocols use some variant of this basic principle. They either differ in the way they define consensus or the definition of deviation from the consensus. Note that these protocols do not require any additional data models than the readings for their functionality. However, the rating assigned to a data reading depends on the number of other “similar” data readings in the set. Thereby, unlike the model-based outlier detection, the rating assigned to a given data reading might change depending on other readings in the set. Our current prototype implementation of RFSN uses consistency based outlier detection because of its generality and applicability to several diverse application contexts.

---

[3] We define the neighborhood of a node as the subset of nodes with which a node can directly communicate by broadcasting a packet

The two most popular approaches for consistency-based outlier detection are distance and density-based outlier detection. Distance-based outlier detection is a very simplistic way of determining the outliers based on the absolute distance between the data readings. The notion of distance-based outliers was proposed in [Knorr and Ng 1998; 1999]. An object p in a data set D is a $DB(x, d_{min})$-outlier if at least x percentage of the objects in D lie at a *distance* greater than d_{min} from p. Here, *distance* represents the difference in the data readings. For example in the case of one dimensional data such as temperature, the distance between two data readings $61F$ and $62.5F$ is equal to the absolute difference between them i.e., $1.5F$. The rating assigned to a data reading using distance-based outlier detection is binary i.e. 0, if it is classified as an outlier, and 1 otherwise. Next we describe density-based outlier detection in more detail, because the current implementation of the RFSN Watchdog uses this mechanism for outlier detection.

3.2.1 *Density-based outlier detection.* Although density-based outlier detection uses the same locality theory as distance-based outlier detection, its consistency definition is more general. Instead of just classifying all the data points in just two clusters, “good” and “outliers”, it allows the data to be classified in as many clusters as possible. Afterwards, each data reading is rated based on two metrics: (1) the relative cardinality of the cluster in which the data point lies compared to the cardinality of the complete data set, and (2) the distance of the data point with rest of the data points in its cluster.

The following example, taken from [Breunig et al. 2000], clearly demonstrates the efficiency of density-based outlier detection. Figure 6 represents a two dimensional data set that contains 502 objects. There are 400 objects in the cluster $C1$, 100 objects in the cluster $C2$, and two additional objects $o1$ and $o2$. In this example, $C2$ forms a denser cluster than $C1$. Intuitively, both $o1$ and $o2$ should be classified as outliers, whereas objects in $C1$ and $C2$ should not. Within the framework of distance-based outliers, either only $o1$ will be characterized as an outlier, or $o1$ and $o2$ as well as all the objects in $C1$ will be classified as outliers. If for every object q in $C1$, the distance between q and its nearest neighbor is greater than the distance between $o2$ and $C2$, then one can prove that there is no appropriate value of x and d_{min} such that $o2$ is a $DB(x, d_{min})$-outlier while the objects in $C1$ are not. A detailed proof can be found in [Breunig et al. 2000]. On the other hand, the density-based outlier detection scheme will classify the data points to lie in two separate clusters, $C1$ and $C2$, whereas both $o1$ and $o2$ will be classified as isolated objects (outliers).

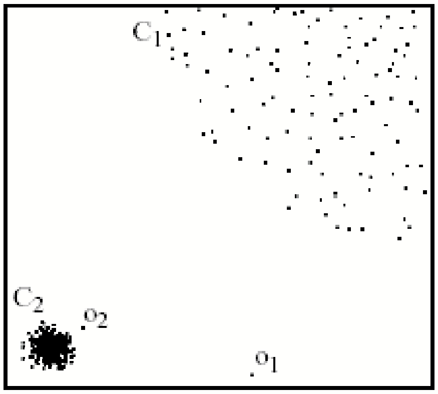


Fig. 6. Representative example scenario showcasing the benefits of density-based outlier detection over distance-based outlier detection
Figure source [Breunig et al. 2000]

Several variants of the density-based outlier detection scheme have been proposed over the past few years [Breunig et al. 2000; Papadimitriou et al. 2002]. These proposals either differ in the way they cluster the data or the way in which they decide the rating for different data points. Our prototype implementation uses the simplest version of density-based outlier detection, which is named Local Outlier Factor, or LOF [Breunig et al. 2000]. This choice was mainly motivated by the fact that LOF consumes the least amount of resources as compared to other density-based schemes such as Local Correlation Integral or LOCI [Papadimitriou et al. 2002], etc. The main advantage of LOCI over LOF is its superior performance for high dimensional data sets. However, since most of the sensor modalities have low dimensions (at most 3), we note that LOF is a good choice for outlier detection in sensor networks.

In the remaining part of this subsection, we will present a brief overview of the LOF algorithm. We refer the readers to [Breunig et al. 2000] for all the details. A key difference between LOF and distance-based outlier detection is that LOF does not enforce outlying to be a binary property. Instead, each data reading is assigned an outlier factor, which is the outlying degree of the reading. The only parameter required by LOF is *minpts*, which corresponds to the minimum number of points that are used to define a neighborhood of a given data reading.

The LOF algorithm works as follows: A virtual circle is drawn around every data reading so that at least *minpts* readings are included in this circle. For example, Figure 7 shows a representative example for reading P and $minpts = 3$. This set of data readings is also referred to as the *minpts-neighborhood* of P. Based on Ps distance to different points in its minpts-neighborhood and the average distance between different readings in this neighborhood, a density parameter is calculated, which is called the *local reachable density* of P. Finally, the local outlier factor, LOF, is calculated by taking the average of the ratio of the local reachable density of P and those of $P's$ minpts-nearest neighbors.

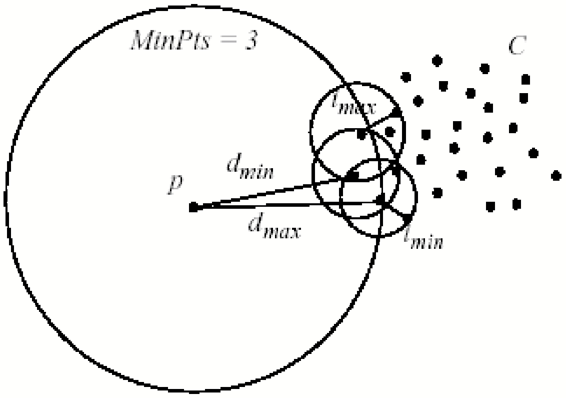


Fig. 7. Local Outlier Factor (LOF)
Figure source [Breunig et al. 2000]

As observed in [Breunig et al. 2000], LOF's performance depends a lot on the choice of *minpts*. Picking an inappropriate value greatly degrades the performance of outlier detection. Thereby, in practice, instead of calculating just one LOF for every reading, we calculate multiple LOF values for different *minpts* varying in the range $(minpts_{lower}, minpts_{upper})$. The final LOF value for a reading is taken as the maximum of all these LOF values. In our prototype implementation, these two parameters, $minpts_{lower}$ and $minpts_{upper}$, are exposed to the applications. We also normalize the LOF values to project them on the scale of (0,1).

3.3 Reputation System

Although reputation systems are widely used in diverse domains, none of the existing systems can be used in the context of RFSN. E-commerce systems, such as ebay [Resnick and Zeckhauser 2000], Yahoo auctions [Resnick et al. 2000], and Internet-based systems such as Keynote [Blaze et al. 1996; Blaze et al.], not only maintain reputation metrics at a centralized trusted authority but also uses a deterministic number for representing reputation. RFSN instead aims to be distributed and statistical in reputation. Much closer to our context are reputation systems designed for ad-hoc networks, Confidant [Buchegger and Boudec 2002] and Core [Michiardi and Molva 2002], and peer-to-peer networks [Xiong and Liu 2003]. These systems are distributed and also maintain a statistical representation of the reputation by borrowing tools from the realms of game theory, which assumes rational behavior by the nodes. As we said in Section 1, there is a mismatch between these other systems and faulty or random misbehavior. RFSN instead borrows tools from decision theory, which assumes the nodes act randomly.

In this section, we describe techniques for estimating a reputation θ based on transactional data. A transaction occurs whenever two nodes make an exchange of information or participate in a collaborative process. With each exchange, the nodes generate ratings indicating the "cooperativeness" of their partner node. For the moment, we consider reputations θ representing the probability that a given node will cooperate when asked to exchange information[4]. Therefore, our reputations θ are contained in the unit interval $[0,1]$, and values of θ closer to one suggest greater cooperativeness. In the next two sections, we discuss a Bayesian framework for updating reputations given the rating from each new transaction. The first approach, based on binary ratings, will be familiar to many readers. The second approach is based on interval ratings and appeals to the Dirichlet Process, which is gaining popularity in the statistics and machine learning literature.

Within this section we address the following topics: *representation* of reputation *update* with new transactions, a *trust* metric as output of the reputation, *integration* of reputation with other information available to a node, and *aging* of the reputation.

3.3.1 *Representation and Update: Binary Ratings.* We begin with a simple binary rating scheme. Suppose a transaction occurs between nodes i and j. Depending on the outcome, the node i will assign the value 1 if node j was cooperative and 0 otherwise[5]. Node i will then update its reputation for node j, incorporating this new data. Independently, node j will create its own rating for the exchange and update its opinion of node i. For simplicity, we will focus on the computations carried out by node i with the understanding that each node in the network will perform similar operations after it completes a transaction.

Let θ denote the reputation of node j held by node i. We adopt a classical beta-binomial framework for estimating reputations [Gelman et al. 2003; Josang and Ismail 2002]. Specifically, we assign to θ a prior distribution $p(\theta)$ that reflects our uncertainty about the behavior of node j before any transactions with i take place. We will take $p(\theta)$ from the beta family, a two-parameter class of distributions which can be expressed as

$$P(\theta)=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \qquad \forall \;\; 0\leqslant\theta\leqslant 1, \alpha\geqslant 0, \beta\geqslant 0 \tag{1}$$

for some choice of α and β, where $\Gamma(\cdot)$ is the gamma function [Gelman et al. 2003]. The mean of a beta distribution with parameters (α,β) is $\alpha/(\alpha+\beta)$ and its variance is $\alpha\beta/(\alpha+\beta)^2(\alpha+\beta+1)$. The beta is chosen, in part, because of its flexibility and ability to peak at any value in the interval $[0,1]$ with arbitrarily small variance [Gelman et al. 2003].

Given θ we then model our binary ratings as Bernoulli observations with success probability θ. That is, let $X\in\{0,1\}$ denote node i's rating of node j for a single transaction. Then, given j's reputation θ, the probability that node j will be cooperative is

$$p(X|\theta)=\theta^X(1-\theta)^{1-X} \tag{2}$$

Once the transaction is complete, we update our reputation using the posterior distribution for θ

---

[4] As we have seen previously, "cooperation" may be thought of either in terms of a node's ability to deliver information or perhaps in terms of the quality of data delivered.

[5] Note that this rating is done in our framework by the Watchdog mechanism, as described in earlier sections.

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int_{[0,1]} p(X|\theta)p(\theta)d\theta} \propto p(X|\theta)p(\theta) \tag{3}$$

In our case, these expressions become

$$\theta^X(1-\theta)^{1-X}\theta^{\alpha-1}(1-\theta)^{\beta-1} = \theta^{\alpha+X-1}(1-\theta)^{\beta+1-X-1} \tag{4}$$

which means the posterior $p(\theta|X)$ again has a beta distribution with parameters $\alpha + X$ and $\beta + 1 - X$. The utility of the choice of a beta distribution is now clear because of its relationship with the bernoulli (binomial) distribution; the beta distribution is the conjugate prior for the bernoulli distribution. Therefore, our reputation framework requires node i to maintain only two parameters to describe the reputation of node j with very simple update rules as each new transaction occurs.

Suppose nodes i and j now conduct n transactions with ratings $X_1, \ldots, X_n \in \{0, 1\}$. Repeating the updates in the previous paragraph, we find that the posterior distribtion for θ after n transactions is again beta with parameters updated as follows.

$$\alpha^{new} = \alpha + n\overline{X} \qquad \beta^{new} = \beta + n - n\overline{X} \tag{5}$$

Therefore, after n transactions, the posterior mean of θ is

$$\begin{aligned} \frac{\alpha + n\overline{X}}{\alpha + \beta + n} &= \frac{\alpha}{\alpha + \beta + n} + \overline{X}\frac{n}{\alpha + \beta + n} \\ &= \frac{\alpha}{\alpha + \beta}\frac{\alpha + \beta}{\alpha + \beta + n} + \overline{X}\frac{n}{\alpha + \beta + n} \\ &= q_n\frac{\alpha}{\alpha + \beta} + (1 - q_n)\overline{X} \end{aligned} \tag{6}$$

where $q_n = (\alpha + \beta)/(\alpha + \beta + n)$ is a probability that tends to zero as $n \to \infty$. This form of the update shows clearly that we are doing a weighted average of the prior mean and the mean of the new observations. The weight on the prior mean goes to zero as the number of new observations grows very large.

3.3.2 *Reputation and Update: Interval Ratings.* Now we describe an update for ratings that are not measured on a binary scale but instead are assigned some value in $[0, 1]$. We can think of these ratings as estimated probabilities, perhaps for the event that a particular data point exchanged between i and j is faulty. Note that the notion of estimated probabilities is much more consistent than binary ratings, when combined with density based outlier detection that was described in the previous section. In this context, we appeal to a slightly more elaborate framework involving Dirichlet processes [Ferguson 1973].

Take $\mathcal{D}(\delta)$ to be a Dirichlet process with base measure δ and let this be our prior distribution. Given observations $X_1, \ldots, X_n \in [0, 1]$, [Ferguson 1973] tells us that posterior is again a Dirichlet process with base measure $\delta(x) + \sum_{i-1}^{n} I_{X_i}(x)$, where I is an indicator of a point mass at the location of the observation X_i. As we will describe in Section 3.3.3, we are ultimately interested in the posterior trust, i.e. the posterior mean of the reputation distribution. When the prior mean is given by μ_δ, the posterior mean of the Dirichlet process is given by

$$q_n \mu_\delta + (1 - q_n)\overline{X} \tag{7}$$

where $q_n = \delta([0,1])/(n+\delta([0,1]))$ tends to zero as $n \to \infty$ and $\mu_\delta = \int x d\delta(x)/\delta([0,1])$ is the mean of the base measure. Suppose we take $\delta([0,1]) = \alpha + \beta$. Then we have

$$q_n = \frac{\alpha + \beta}{n + \alpha + \beta} \tag{8}$$

which, even though we now are dealing with real-valued observations on the interval $[0,1]$, gives the same weights as in Section 3.3.1, where we had binary cooperativeness ratings. In fact, in order to match not just the the weights q_n but also the prior mean μ_δ, we could take our measure δ to be $(\alpha+\beta)\mathrm{Beta}(\alpha, \beta)$ and get exactly the same updating as in Equation 6 with real-valued variables $X_1, \ldots, X_n \in [0,1]$ instead of binary variables.

Once we have seen that the update is of a generalizable form using the Dirichlet Process, we can also see that the update using binary ratings in Section 3.3.1 can also be derived within this framework. If we let the measure $\delta = \beta I_0 + \alpha I_1$, which would suggest our data are binary, then the update for the mean is again exactly Equation 6. We can now see that this justification is a very general one.

Following from this discussion, in order to maintain our two parameters α, β in a way so that we correctly update the posterior mean, we replace the bayesian update step with an identical bookkeeping step. After a single transaction, if the assigned probability of cooperativeness were $p \in [0,1]$, the beta parameter updates would be

$$\alpha^{new} = \alpha + p \qquad \beta^{new} = \beta + 1 - p \tag{9}$$

3.3.3 *Trust.* The main objective of the reputation block is to expose an output metric that can be used as a representative of the subjective expectation of the other node's future behavior. Up until now we have represented i's reputation of node j with θ, but from here on we represent it with R_{ij} to make the pairwise reputations more explicit. Given a reputation metric R_{ij}, we define the *trust metric* T_{ij} as node $i's$ prediction of the expected future behavior of node j. T_{ij} is obtained by taking a statistical expectation of this prediction:

$$T_{ij} = E[R_{ij}] = E[Beta(\alpha_j, \beta_j)] = \frac{\alpha_j}{\alpha_j + \beta_j} \tag{10}$$

This trust metric can be used by a node in several ways. Some notable ones are:

(1) *Data Fusion*: T_{ij} can be used as a weight for a data reading reported by node j. The data fusion can be then performed on these weighted data readings, thereby reducing the impact of untrustworthy nodes.

(2) *Node revocation*: The evolution of trust over time provides an on-line tool to the end-user to detect compromised or faulty nodes. This can help the end-user to take appropriate countermeasures such as replacing the misbehaving node or sensor.

(3) *Decentralized decision making*: In a heterogeneous sensor network, different nodes might be equipped with different capabilities. For example, a few of

them might have a more precise temperature sensor or a camera, others may be mobile etc. Given a requirement of using a particular service from some other node in the network and faced with multiple choices, the value of T_{ij} can be used as a decision making criteria.

We will elaborate on some of these uses of the trust metric in the context of real world sensor networking applications in Section 5.

3.3.4 *Reputation Integration.* Every node builds upon its reputation table by making direct observations about other nodes in the neighborhood through the Watchdog mechanism. Another way of learning about other nodes in the network is to make use of the experiences of other nodes in the neighborhood. Different nodes have different reputations for other nodes because they may have developed the reputation based on a disjoint set of events. For example, a failure may not span an entire node neighborhood; perhaps a packet containing a data reading is broadcast but is only received at a subset of nodes in the neighborhood. Thus, node i could develop a reputation of node j not only on the basis of its own direct observations but also indirectly through another node k (which also has developed some reputation of node j). We classify these indirect observations as *evidence* and this step as *reputation integration.*

We would like to point out that the objective of the integration step is not to establish consistent reputation metrics at all the nodes. Our only objective is to make nodes share experiences (via reputation). This can be conceptualized as external evidence for a node to establish more concrete reputation metrics. This is useful for several reasons: (1) propagating reputation information in the network facilitates the formation of the community of trustworthy nodes, (2) convergence time is shorter than when relying solely on direct observations, (3) acquiring information through external evidence is much cheaper from an energy perspective, as you get information about several nodes in a single packet (observation), and (4) this allows nodes to learn about distant nodes (not in their neighborhood), which it cannot monitor directly. We now describe the integration step, followed by potential for abuse introduced by integration.

Node i receives reputation information about node j through node k. Let us represent these indirect observations by (α_j^k, β_j^k). Node i already have prior reputation information about j and k, represented by (α_j, β_j) and (α_k, β_k) respectively. We need to combine these pieces of information to obtain a new reputation metric for j, $(\alpha_j^{new}, \beta_j^{new})$. The simplest strategy is to treat the new reputation information received from node k and the new observations made by node i about node j using the Watchdog mechanism equivalently. Following this, the new reputation metric can be derived as follows:

$$\alpha_j^{new} = \alpha_j + \alpha_j^k ; \beta_j^{new} = \beta_j + \beta_j^k \tag{11}$$

However, this simplistic strategy exposes some potential avenues for abuse, such as “Bad-mouthing” and “Ballot stuffing” attacks [Buchegger and Boudec 2003b; Dellarocas 2000]. E-commerce systems such as e-bay and yahoo auctions have been specifically found to be highly vulnerable to “bad-mouthing attacks”, in which sellers can collude with buyers to drive other sellers out of the market [Dellarocas 2000]. The conspiring buyers provide unfairly negative ratings to the targeted sellers, thereby lowering their reputation. Reputation-based systems developed for ad-hoc networks such as Confidant try to limit the damage of these attacks by relying on trusted nodes to report bad behavior. They use special messages called alarm messages to handle this case [Buchegger and Boudec 2003a]. However, this makes the system vulnerable to retaliation attacks, as analyzed in [Michiardi and Molva 2003]. In an e-commerce system sellers can also collude with buyers, in order to obtain unfairly high ratings from them [Dellarocas 2000]. This will have the effect of inflating a seller's reputation, thereby allowing the seller to receive more orders. This kind of "ballot stuffing" attack can also be achieved in distributed reputation-based systems such as Core and Confidant.

It can be noted that a malicious adversary can carry out both bad-mouthing and ballot stuffing attacks on the simplistic integration strategy. Intuitively, node i should give more weight to the direct observations made by it than the evidence obtained from other nodes. Furthermore, the evidence from different nodes should be weighted in ratio of their respective reputation. Based on these observations, we use an approach, proposed in [Josang and Ismail 2002], that is based on the concept of belief discounting [Josang 2001]. In this approach, the problem is mapped into an equivalent problem in the realm of Dempster-Shafer belief theory [Shafer]. We refer the readers to [Josang and Ismail 2002] for all the details. The closed form expressions for the new reputation metric are given by the following equations:

$$\alpha_j^{new} = \alpha_j + \frac{2\alpha_k \alpha_j^k}{[(\beta_k + 2)(\alpha_j^k + \beta_j^k + 2)] + 2\alpha_k} \tag{12}$$

$$\beta_j^{new} = \beta_j + \frac{2\alpha_k \beta_j^k}{[(\beta_k + 2)(\alpha_j^k + \beta_j^k + 2)] + 2\alpha_k} \tag{13}$$

Let us do some simple sanity checks on these equations. Consider the special case of an untrustworthy node, i.e. $\alpha_k << \beta_k$. In this scenario, the two equations can be approximated as follows:

$$\alpha_j^{new} = \alpha_j + \frac{2\alpha_k \alpha_j^k}{\beta_k(\alpha_j^k + \beta_j^k + 2)}; \beta_j^{new} = \beta_j + \frac{2\alpha_k \beta_j^k}{\beta_k(\alpha_j^k + \beta_j^k + 2)} \tag{14}$$

In case of a bad-mouthing attack, node k will propagate $\alpha_j^k << \beta_j^k$ and hence, the two equations can be further reduced to:

$$\alpha_j^{new} = \alpha_j + \frac{2\alpha_k \alpha_j^k}{\beta_k \beta_j^k}; \beta_j^{new} = \beta_j + \frac{2\alpha_k}{\beta_k} \tag{15}$$

However, as $\alpha_k << \beta_k$, the end result of the equations will be: $\alpha_j^{new} \sim \alpha_j; \beta_j^{new} \sim \beta_j$. Thus, an untrustworthy node will not be able to perform a bad-mouthing attack. In case of a ballot stuffing attack, node k will propagate $\alpha_j^k >> \beta_j^k$ and hence, the two equations can be further reduced to:

$$\alpha_j^{new} = \alpha_j + \frac{2\alpha_k}{\beta_k}; \beta_j^{new} = \beta_j + \frac{2\alpha_k \beta_j^k}{\beta_k \alpha_j^k} \tag{16}$$

However, as $\alpha_k << \beta_k$, the end result of the equations will be: $\alpha_j^{new} \sim \alpha_j; \beta_j^{new} \sim \beta_j$. Thus, an untrustworthy node will not even be able to perform a ballot stuffing attack. Furthermore, it can be verified that if two nodes perform the same evidence, the effect on the eventual reputation ($\alpha_j^{new}, \beta_j^{new}$) gets weighted in ratio of their respective reputations.

3.3.5 *Aging.* We incorporate an aging mechanism in RFSN so that a node's trustworthiness is reevaluated continuously. This provides resiliency against *sleeper attacks*, often witnessed in the context of e-commerce systems, wherein an entity behaves accurately for some stipulated amount of time, creates a good reputation for itself, and then starts misbehaving. In this mechanism, the reputation metrics of the node are decreased periodically by a weight w as follows:

$$\alpha_j^{new} = w * \alpha_j; \beta_j^{new} = w * \beta_j \tag{17}$$

The two parameters required by the aging mechanism are the periodic rate at which the aging is performed and the weight w. Our prototype implementation of the RFSN middleware service exposes both these parameters to be configured by the application developers. The aging period can be set to the average interval between successive observations, but choosing an appropriate value of w is not straightforward. In [Buchegger and Boudec 2003a], authors provide a technique for finding out this weight by comparing the evolution of the reputation in a system with and without aging weight respectively. We refer the readers to [Buchegger and Boudec 2003a] for the details. In Section 5, we will demonstrate the efficacy of this aging procedure through a specific example.

4. MIDDLEWARE SERVICE ON MOTES

RFSN is available as a middleware service on Motes. It is currently supported in two operating systems, TinyOS and SOS. In this section, we explain in detail the prototype implementation that has been developed for SOS. We will provide an evaluation of this service in context of a temperature monitoring sensor networking system in the Section 5.

4.1 Overview

Applications in SOS are written as a collection of interacting modules. Our prototype implementation of RFSN consists of two modules: (1) The Watchdog module implements the outlier detection schemes, and (2) The Reputation module implements reputation representation, updates, integration and trust evolution. Following a brief description of SOS in Section 4.2, we explain both these modules and the APIs that are exposed by these modules to the applications in Sections 4.3 and 4.4. Next in Section 4.5, we discuss the memory and energy overhead in the context of a data collection application. Finally we describe the graphical front end interface for RFSN in Section 4.6.

4.2 SOS Background

SOS [Han et al. 2005] is an operating system for resource constrained embedded sensor nodes. It consists of a thin kernel that is statically installed on all the nodes in the network. The rest of the system and the application components are implemented as modules, for which the SOS kernel provides support for run-time dynamic loading and unloading. In addition, the kernel implements mechanisms for inter-module communication, as well as a rich set of services such as dynamic memory allocation, software timers, sensor management and high-level I/O interface. The kernel services are accessible by the dynamic modules via an extra level of indirection through a jump table. Applications in SOS are a collection of interacting modules. Inter-module communication in SOS can be synchronous or asynchronous. Asynchronous communication is implemented using message passing. Each module is implemented as a message handler that processes messages. Modules post messages, which are queued by the SOS kernel and dispatched to the corresponding destination module. Synchronous communication between the modules is implemented using dynamic linking. Every module indicates the set of functions that it provides to the rest of the system and functions to which it subscribes from the rest of the system. The dynamic linker tracks down and links all the provide-subscribe function pairs when each module is loaded. Therefore, a module in SOS just needs to define its synchronous and asynchronous interfaces, and afterwards it can be combined with other modules to form a valid application.

4.3 Watchdog module

The Watchdog module maintains a library of outlier detection protocols. It currently implements two protocols - distance-based outlier detection and LOF (density-based outlier detection), as described in Section 3. The module provides two asynchronous interfaces (message types): *msg-distance* and *msg-density*, for using the distance- and density-based outlier detection respectively. The calling application specifies both its identity and the asynchronous interface on which it expects a reply from the Watchdog module. The calling application is also responsible for providing the requisite parameters for the corresponding outlier detection mechanism. This provides the applications with the full freedom to appropriately customize the working of the protocols to their specific needs.

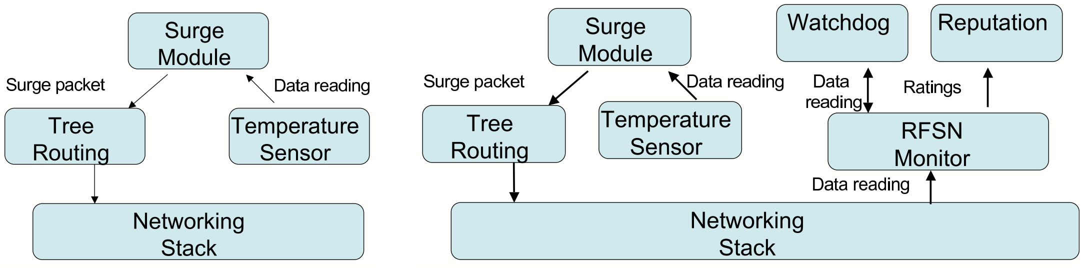
The Watchdog module has been written in a manner so that it can be easily populated in the future with more outlier detection protocols. In order to add a protocol X to this module, the user needs to perform the following tasks: (1) Add a new message type, *msg-X*, (2) Add a new data structure containing the parameters that are required by the protocol X, and (3) Provide the C-code function that implements this protocol.

4.4 Reputation module

This module maintains the reputation metric, as requested by different applications, in the following data structure.

```
struct { uint32_t context; rfsn_param_t param; uint8_t size;
    reputation_t* reputation; } rfsn_t;
```

`context` acts as the identifying tag for the given reputation metric. This information is provided by the applications when a new reputation metric is initialized, and from then onwards every time for a reputation update. `size` holds the total number of reputations maintained in a given reputation metric. The three parameters of RFSN (aging period, aging weight and integration period) are stored in `rfsn_param_t` and the reputation parameters (α, β) along with the node identities are stored in `reputation_t`.



(a) Surge (b) Surge-RFSN

Fig. 8. Data Monitoring and Collection Application in SOS

The Reputation module exposes three asynchronous interfaces (message types): *msg-reputation-init*, *msg-reputation-update*, and *msg-param-update* for reputation initialization, updates and RFSN parameter updates, respectively. The reputation updates, aging, integration and trust evolution are implemented as described in Section 3. Reputation integration takes place under the hood without the knowledge of the applications. Periodically, the Reputation module broadcasts a packet that contains the reputation information of nodes in the neighborhood. The period is equal to the integration period as specified by the applications. The Reputation module also provides a synchronous function call that returns the trust metric of a given node.

4.5 Complexity in Context of a Data Monitoring Application

In this section, we evaluate the implementation complexity and the energy/memory overhead of the RFSN middleware service in the context of a general data monitoring application. A data monitoring application forms the basic building block of several sensor networking systems. SOS provides a prototypical data collector and tree builder application. In the sensor networking literature, this complete application is referred as Surge. The implementation also involves a module called surge, which we do not capitalize for clarity. The main components of this application are shown in Figure 8(a). The surge module periodically samples the desired sensors and uses tree routing to transmit the data to the base-station. The surge module, tree routing and the different sensor drivers such as temperature, photo, acoustic etc. are implemented as binary modules in SOS.

Although Surge provides a stable implementation of a data monitoring application, it does not provide any mechanisms to verify the validity of the sensor data. As a result, sensor network applications that are based on Surge are highly vulnerable to both malicious attacks and non-malicious system faults that can corrupt the sensor data. As shown in [Wagner 2004], a single corrupted sensor can drastically change the aggregation metrics, such as average, maximum, minimum etc.

4.5.1 *Secure Data Monitoring Application.* We have enhanced the Surge application to make it resilient to non-malicious sensor faults by adding the RFSN middleware service. We name this application as Surge-RFSN. The components of the Surge-RFSN application are shown in Figure 8(b). In this application, every node also maintains a reputation metric for all the other nodes in its neighborhood using the RFSN-monitor, Watchdog and Reputation modules. In every time epoch, besides sampling and transmitting the data, each node collects the temperature readings from the other nodes in the neighborhood. This data array, which corresponds to the readings in the same epoch, is passed to the Watchdog module. Density-based outlier detection is used to assign a rating to each data reading, which is used to update the reputation for every node accordingly. In the previous section, we have explained the functionality of the reputation and Watchdog module: the RFSN-monitor module collects the readings from all the other nodes in the neighborhood and calls the interfaces that are provided by the Reputation and Watchdog modules appropriately. RFSN-monitor exploits the broadcast property of the communication medium. When a node transmits a packet to its parent in the hierarchical tree structure, other nodes in the neighborhood can also passively listen to this packet. As a result, no additional packets are transmitted in Surge-RFSN as compared to Surge for maintaining reputation for every node. It is important to note that the part of Surge-RFSN application that collects the data and transmits it to the base-station using tree routing is the same as those in the Surge application. We have not made any changes in either surge, tree routing or the data sensor modules. This clearly demonstrates the ease with which RFSN can be incorporated in existing sensor network applications. It does not impose any change in the existing application software.

RFSN was designed with the goals of developing a distributed, localized and a lightweight reputation system for sensor networks. We visualize each node in the network to be running the software framework, equivalent to the block diagram shown in Section 3. In a centralized data collecting application such as Surge, where the final results are only aggregated at the root node, it would have been fine even if we ran the RFSN middleware only on the root node. In general, it will depend on the nature of the application, as to where this service should be run. RFSN is flexible and can be customized to be run in a centralized manner for applications such as Surge or in a distributed manner for distributed applications such as event monitoring. In the following subsections, we analyze the overhead of running the RFSN middleware at a node, so that it can be used in the context of both centralized and distributed operation of RFSN.

4.5.2 *Memory overhead.* Table I shows the memory footprint of the different components in Surge-RFSN and Surge respectively. The Surge application (SOS kernel + surge module + Tree routing + Temperature Sensor) takes 40552 bytes of ROM, whereas the Surge-RFSN application (SOS kernel + surge module + Tree routing + Temperature Sensor + RFSN-monitor + Reputation + Watchdog) takes 47274 bytes of ROM; the memory overhead of Surge-RFSN is around 6Kb of ROM. As can also be observed from Table I, the static RAM memory allocation for the complete RFSN middleware service (RFSN-monitor + Reputation + Watchdog) is only 18 bytes. We note that this does not include the memory used for storing the data readings by the RFSN-monitor nor does it include the reputation metrics by the Reputation module. The memory needed to maintain this state information is allocated dynamically by the respective modules using the dynamic memory allocation API provided by the SOS Kernel.

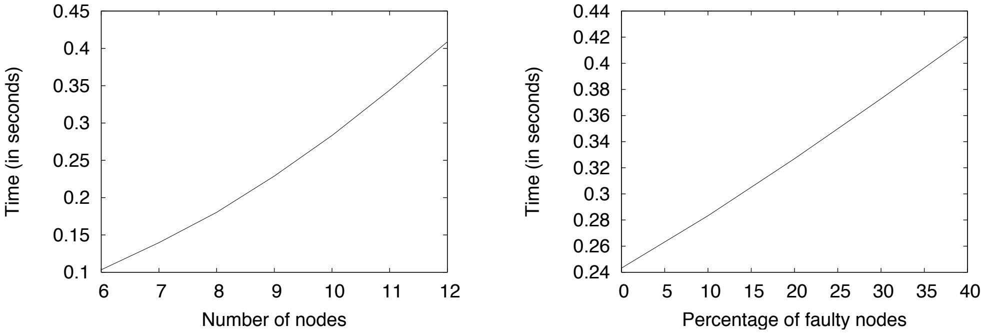
Component	ROM (in bytes)	RAM[6] (in bytes)
surge module	578	11
Tree routing	2080	10
Temperature sensor	174	0
RFSN-monitor	934	12
Reputation	2230	6
Watchdog	3558	0

Table I. Memory footprint of the different SOS components

4.5.3 *Energy overhead.* The extra energy overhead of the Surge-RFSN application can be due to the following three reasons. First, the Reputation module transmits packets that contain the reputation information (integration step). As mentioned in Section 3, the period of this integration can be configured by the applications. In Surge-RFSN we have configured this parameter to -1, which indicates the Reputation module does not perform any reputation integration. Our objective is to reduce the extra energy overhead of Surge-RFSN. All the results that are shown in the next section correspond to this configuration. As we will show, even without any reputation integration, Surge-RFSN can obtain considerable benefits over Surge.

In both Surge and Surge-RFSN, a node is at least involved in the following two types of data communication: (1) The node transmits a packet containing its data reading to its parent node in the hierarchical tree structure, and (2) It collects the data readings from all its children, provided that it is not a leaf node, and forwards them towards the base-station. In addition to this, Surge-RFSN keeps the radio in the promiscuous mode to snoop on the data communication, in which it is neither involved as a parent nor as a child. In Surge, the radio will be in idle mode after completing these two tasks. Since the idle mode and receive mode power consumption are comparable for typical sensor networking platforms, the extra energy spent in snooping on the data packets in case of Surge-RFSN is negligible. In [Szewczyk et al. 2004], the authors have coupled Surge with BMAC [Polastre et al. 2004], a duty-cycling MAC protocol, wherein the radio is put to sleep mode during the time of inactivity. This results in a significant decrease in the energy consumption. If all the data communication in a neighborhood takes place during a small fraction of the time epoch, as in [Szewczyk et al. 2004], a similar strategy can also be developed for Surge-RFSN.

The last and perhaps the foremost reason for increased energy consumption of Surge-RFSN is the additional CPU utilization in performing the density-based outlier detection and updating the reputation of the nodes accordingly. As reputation update is a simple addition operation, the key bottleneck becomes the density-based outlier detection. Our objective was to profile the time needed by a Mica2 mote in executing LOF. This time profiling was conducted using Avrora [Titzer et al. 2005]. Avrora provides a cycle-accurate simulation of the AVR microcontroller, allowing real programs to be executed with precise timing. In addition, it also provides analysis and profiling tools to gauge the program cycle count. Avrora provides a flexible framework to set up profile points at any places in the code. Using these profile points, the accurate cycle count can be derived for any code snippet. In our scenario, the profile points were set to the beginning and end of the density-based



(a) Time taken in running LOF for different neighborhood sizes. In every scenario, only one of the nodes is faulty.

(b) Time taken in running LOF for different percentage of faulty nodes in the neighborhood. Total number of nodes is equal to 10.

Fig. 9. Time complexity of density-based outlier detection (LOF) on motes

outlier detection respectively. The time measurement was derived by multiplying the cycle count with the frequency of the atmega128 microcontroller ($\sim 8Mhz$).

The time taken is a function of the neighborhood size as well as the percentage of faulty nodes within the neighborhood. Figure 9(a) plots the time needed for varying neighborhood sizes, whereas Figure 9(b) plots the time needed for a fixed neighborhood size of 10 but for a varying fraction of faulty nodes. The most interesting observation is the scale of the y-axis in both Figure 9(a) and Figure 9(b). Even for a neighborhood sizes of 12 and even when the percentage of faulty nodes was the maximum allowed (<50%), the time taken in running the density-based outlier detection is under a second.

To put into perspective, this energy overhead of LOF can only be evaluated vis-a-vis the frequency with which it will be run. LOF will be run whenever a new observation is taken in a data collecting application or whenever a new event happens in a data monitoring application. This will be typically of the order of a few minutes in a data collecting application, a representative real-life example is discussed in the next section, and can be up to a few hours in a monitoring application. Given this, the energy overhead of an extra second of CPU processing every few minutes will be negligible.

Thereby, overall the energy overhead due to the additional CPU utilization of running the RFSN middleware service is negligible.

4.6 Graphical front-end

RFSN can help us identify the misbehaving nodes in real time, while the network is in operation. In order to supplement this functionality, we have developed a real-time visualization tool. This user interface is provided through an Apache web server. The visualization tool can be run either on a desktop, laptop or even on a handheld IPAQ. Every sensor node is configured to periodically transmit a *visualization tool* packet that contains the trust metrics of the other nodes in the network that it has established so far. The graphical front-end can either receive these packets directly, if it also equipped with a corresponding radio interface (for example CC1010 radio for receiving packets from Mica2 nodes), or it can also receive them over the UART from a specific sensor node. As we will show in the next section, in our lab scale testbed, an Apple laptop acts as the GUI front-end, and it is connected to one specific node through the UART. After receiving these packets, the GUI front-end parses them, extracts the reputation information, and stores it in a temporary database. Each database entry is in the form of a triplet, (source-id, node-id, trust metric), which represents the trust metric formed by the node, source-id, about the neighboring node, node-id. To realize a graphical user interface that reflects this reputation information in real-time, we use a mechanism that combines perl/cgi script (runs on web server) and javascript (runs on browser client). Each time a user browses the perl/cgi script that executes on the GUI front-end, the script triggers the GUI front-end to read database files that contain the reputation information. This information is transformed to a javascript which is transmitted back to the user. In this way, we achieve a graphic visualization of the trust metrics. The real-time update of the reputation information is achieved by the javascript's mechanism that reloads the perl/cgi script periodically.

5. EVALUATION

In this section, we evaluate the performance of the RFSN middleware service in identifying misbehaving nodes using multiple contexts and experiments. First, we describe an evaluation performed on a lab-scale temperature monitoring system consisting of Mica2 Motes. In order to gather sufficient statistics, we repeated the experiments (same network topology and similar sensor data) on simulations using Avrora [Titzer et al. 2005]. We deliberately fabricate faults at one of the nodes in the experiments. The taxonomy of the faults has been adopted from our own preliminary work in [Balzano]. We do not claim this list to be exhaustive; albeit it is a good representation of the faults that have been witnessed in existing sensor network deployments. Next, we describe an evaluation performed on data collected from the James Reserve. In this way we demonstrate the usefulness of RFSN on real-world sensor network application data.

5.1 Lab Experiments

First, we explain the experimental setup and the system model.

Lab-scale testbed The lab-scale testbed consists of five motes, numbered 1 to 5. Each one of these motes was equipped with a MTS300 sensor board to take a temperature reading in every time epoch. The duration of the time epoch was set to 5 minutes. These motes were placed in the center and the four corners of a 6' x 6' room. All the motes lie in the direct communication range of each other. Each of these motes was compiled with exactly the same Surge-RFSN application that consisted of the SOS kernel and the modules: surge, tree routing, temperature sensor driver, rfsn-monitor, reputation and watchdog. The mote numbered 1, which lies in the center of the room, was connected to an Apple Laptop through the UART. In addition to performing exactly the same steps as rest of the nodes, mote 1 was configured to tunnel the reputation metrics that it has established for the other nodes to the UART in every time epoch. The Apple laptop was running the GUI application that was discussed in the previous section. This allowed us to visualize the trust metrics that mote 1 has established for rest of the motes, 2-5, in real time.

In order to gauge the efficacy of Surge-RFSN, we emulated a vast variety of sensor faults at the mote numbered 5. This emulation was done by adding a data filter in the temperature sensor driver (only at mote numbered 5), wherein the readings from the ADC were manipulated to follow the desired data model. Our objective was to find out whether the reputation metric for this node at mote 1 differed from the other motes, that were numbered 2-4. The following subsections summarizes the results that were obtained for different types of faults.

Avrora simulations In order to gather sufficient statistics for accurately gauging the performance of RFSN under the impact of different fault models, we use simulations in Avrora [Titzer et al. 2005]. Avrora provides a cycle-accurate simulation of the AVR microcontroller, allowing real programs to be executed with precise timing. One of Avrora's main features is that the same embedded code that runs on Mica2 Motes can be directly executed by the simulator, thus leaving little possibility of experimental errors in moving from real world setting to simulations. In fact, we were able to recreate the same network topology and were using the same embedded code as in the lab-scale temperature monitoring system. The only hurdle is that Avrora cannot simulate the physical world and hence, we had to emulate the temperature readings measured by the different sensors. For the non-faulty nodes, numbered 2-4, the sensor reading during any time epoch was set to 25 + calibration error. In one of our previous works [Jew et al.], we had performed a detailed experimental study to find out the calibration error model for temperature sensor on the MTS300 board. We adopt the same calibration error model, whereby it is modeled using a Gaussian distribution, $N(0, 2)$. The reading for the faulty node, mote 5, was emulated in accordance with the corresponding fault model. The statistics using Avrora simulations are averaged over 100 independent experimental runs.

RFSN parameter settings As discussed in Section 3, the three parameters required by the RFSN middleware are: aging weight, aging period and integration period. In all our experiments, we fix the aging weight to be equal to 0.95. The aging period is fixed to the duration of a time epoch, equal to 5 minutes. No reputation integration is performed in any of these experiments, i.e., integration period is set to -1.

5.1.1 *Offset fault.*

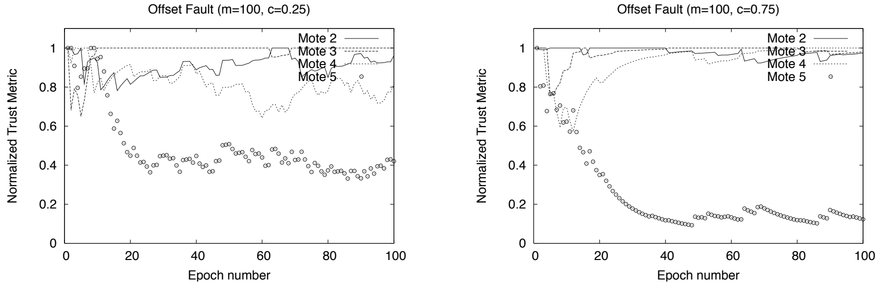
The offset fault most commonly manifests itself as an additive constant on top of the pre-fault measurement value and is represented as follows:

$$f = x + m \quad with \quad probability \quad c \tag{18}$$

$$f = x \quad with \quad probability \quad 1 - c \tag{19}$$

Here x represents the pre-fault value measured by the temperature sensor. Note that x also includes the possible calibration error from the true value of the ambient temperature. The faulty value, f, is obtained by adding the offset m, with probability c, to the pre-fault value, x. Figure 10 plots the evolution of trust for the 4 motes in the presence of offset faults at mote 5, as observed at the Apple laptop acting as the GUI frontend. Note that we introduce the fault at mote 5 after the 10th epoch.

As it can be observed from Figure 10, the trust metric for mote 5 is significantly different from the trust metrics of the rest of the nodes. Furthermore, the trust metric is relatively higher for mote 5, when the offset faults happen with lower



(a) m = 100, c = 0.25 (b) m = 100, c = 0.75

Fig. 10. Trust evolution in presence of offset faults

Fault Model	Node 2	Node 3	Node 4	Node 5
m=100, c = 0.25	0.95	1.0	0.9	0.42
m=100, c = 0.75	0.99	1.0	0.97	0.12
m=100, c = 0.5	0.99	1.0	0.94	0.23
m=10, c = 0.5	0.98	1.0	0.99	0.25
m=5, c = 0.5	0.99	1.0	0.97	0.26
m=2, c = 0.5	1.0	0.9	0.95	0.68

Table II. Trust evolution in presence of offset faults

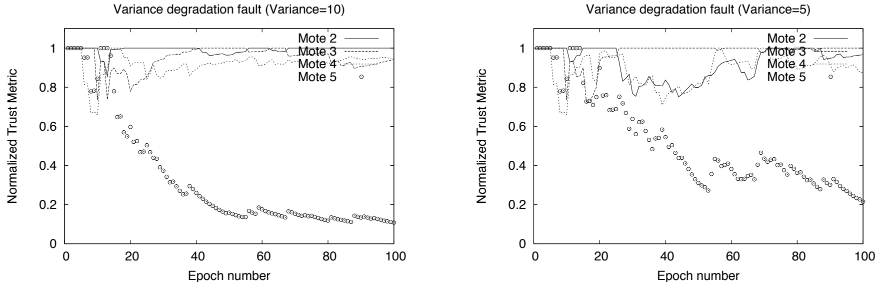
probability, $c = 0.25$ as compared to $c = 0.75$. We repeated the experiment in Avrora simulations with several other settings of m and c. Table II shows the final trust metrics that were established for the 4 motes after 100 epochs. The values in the table have been obtained after averaging over 100 independent runs. As can be observed from Table II, the trust metric degrades, almost linearly, with the likelihood of mote 5 being faulty (c). However, a keen observation to be made is that the value of m, as far as it is not too small for classifying mote 5 to be faulty (for example when $m = 2$), does not have much impact on the final trust metric of mote 5.

5.1.2 *Variance Degradation Fault.* The variance degradation fault is a fault where over time the sensors becomes less and less accurate. The variance of the reported measurement is much higher than that of other new sensors. This fault can be represented as follows:

$$f = x + N(0, \sigma) \tag{20}$$

Note that the mean value of this faulty sensor, over large durations of time, will be the same as an accurate sensor, but its value cannot be trusted with the same magnitude. Figure 11 plots the evolution of trust in this scenario for two different values of σ. In both cases, motes 2-4 are able to establish significantly higher trust metrics than mote 5.

As can be observed from Table III, the trust metric for mote 5 is higher for a lower value of variance and vice-versa. Note that $\sigma = 2$ corresponds to the case when all the motes are indistinguishable from one another. This is because we have configured the non-faulty nodes, $2 - 4$, to have a slight variance in there readings

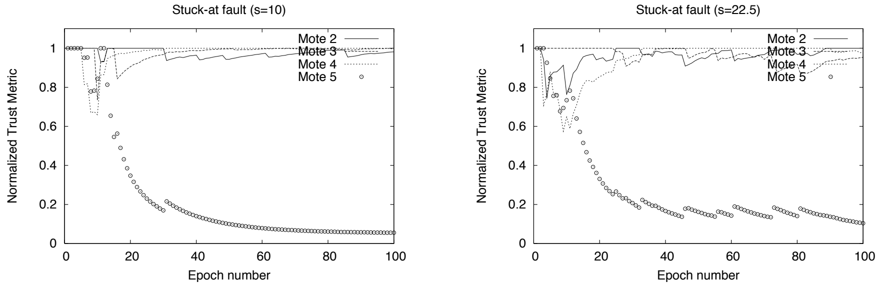


(a) Variance = 10 (b) Variance = 5

Fig. 11. Trust evolution in presence of variance degradation faults

Fault Model	Node 2	Node 3	Node 4	Node 5
$\sigma = 2$	0.95	1.0	0.98	0.97
$\sigma = 5$	1.0	0.93	0.89	0.34
$\sigma = 10$	1.0	0.952	0.964	0.18
$\sigma = 20$	0.99	1.0	0.99	0.11

Table III. Trust evolution in presence of variance degradation faults



(a) s = 10 (b) s = 22.5

Fig. 12. Trust evolution in presence of Stuck-At faults

due to calibration error, which is also equal to 2.

5.1.3 *Stuck-At Fault.* The stuck-at fault represents a sensor getting stuck at a particular value s, i.e.

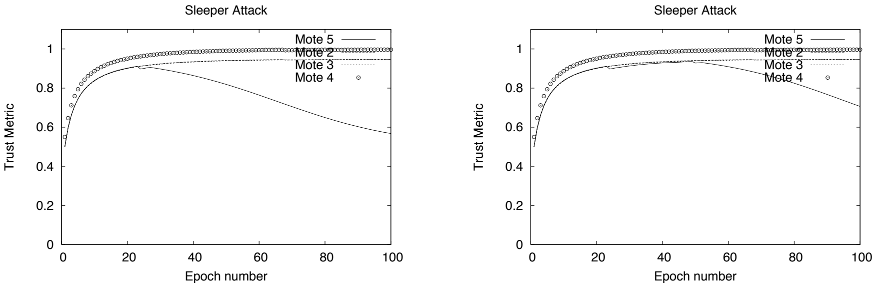
$$f = s \forall x \tag{21}$$

Figure 12 plots the evolution of trust for the 4 motes in the presence of variance degradation faults at mote 5. As can be observed from Figure 12, mote 5 can be easily classified as a faulty node in these two scenarios.

The stuck-at faults can be very dangerous depending on the value of s. If this value lies within the sensing range, it might be very hard to discover such a fault. Table IV summarizes the trust metrics for different values of s. As the values of the non-faulty nodes also have a mean value of 25.0, we were unable to distinguish

Fault Model	Node 2	Node 3	Node 4	Node 5
s = 10	0.98	0.99	1.0	0.06
s = 20	0.99	0.99	1.0	0.08
s = 22.5	1.0	0.97	0.99	0.1
s = 25.0	1.0	0.97	0.99	0.9

Table IV. Trust evolution in presence of stuck-at faults



(a) Stop sending after 25 epochs (b) Stop sending after 50 epochs

Fig. 13. Trust evolution in presence of Sleeper attacks

mote 5, when $s = 25.0$.

5.1.4 *Sleeper attacks.* In a real world deployment, a node can become completely non-functional after some time and stop reporting any values. Note that this is different from reporting a zero reading, which can be classified as a stuck-at fault. In this scenario, the node does not transmit any data reading, as if it does not exist in the network. Figure 13 plots the evolution of trust in this scenario when mote 5 stops reporting after 25 and 50 epochs respectively. Note that unlike the previous subsections, we plot the absolute value of trust metrics (and not the normalized trust metrics) in Figure 13.

As it can be observed from Figure 13, the four motes are completely indistinguishable until the time interval when mote 5 stops reporting. Afterwards, the trust metric of mote 5 degrades continuously. This is the result of the continuous aging of the reputation metrics, using aging weight and aging period.

5.1.5 *Summary.* The experimental study in the above section demonstrates the efficacy of RFSN in identifying misbehaving nodes for a variety of fault scenarios. However, we would like to point out that the current outlier detection mechanisms do not provide an effective countermeasure against all possible malicious attacks and faulty scenarios. Consensus-based schemes inherently rely on the redundancy of sensor data information in the system. We note that there are several ongoing research efforts that deal with the design of complex outlier detection schemes. The notable ones are: (1) In [Koushanfar et al. 2002; 2003; Koushanfar and Potkonjak 2005], the authors have developed a suite of model based outlier detection protocols for identifying the misbehavior of the light sensing modality, and (2) In [Ramanathan et al.], the authors have developed a comprehensive outlier detection strategy for their soil monitoring sensor networking system. Although these schemes can be extremely useful, none of them provide a complete system that maintains the misbehaving information of the nodes over time and uses it in future to make in-network decisions. The RFSN middleware service provides this generalized framework. While designing the RFSN service, we have taken special care to make it flexible enough to allow the integration of other outlier detection schemes. Schemes such as [Koushanfar et al. 2002; 2003; Koushanfar and Potkonjak 2005; Ramanathan et al.] can be easily integrated in the Watchdog of the RFSN middleware to provide a complete reliable operation of a sensor networking system.

5.2 Sensor Data Experiments

In this section, we evaluate the performance of the RFSN using real data sets collected from the James Reserve deployment. Our objective, in this subsection, is to demonstrate the reputation that would have been established for different nodes, had they been using RFSN. We show RFSN could be instrumental in identifying misbehaving nodes or sensors. To show this, we took two sensor data traces collected at a James Reserve deployment. After this, we replayed this trace in simulations and enabled the motes to use the RFSN middleware service for establishing reputation for other neighboring nodes. These simulations were done in Avrora. We note that Avrora provides a tool for doing trace-based simulations, wherein the readings measured by any sensor can be read from a file. Furthermore, Avrora also allows us to create exactly the same spatial and temporal scales in these simulations as the real deployment.

5.2.1 *Measuring the Cold Air Drainage.* The first representative example comes from sensor network deployment done in a valley at James Reserve. In this application, motes are collecting temperature measurements over time and space. The scientists are interested in a cold air drainage event, which is a moment before dawn when cold air rushes down a valley to the lowest point. The motes are placed perpendicularly to the cold air front. Currently, each data point is of interest to the scientsts and thus is transmitted to a sink. However, the average temperature also reveals the cold air drainage event, and this single value could be sent to save energy when the nodes are deployed in a longer lived or more remote deployment.

In all, the deployment consists of 26 motes, each taking a new reading approximately after every 4 seconds. Of these, readings were collected from all but the two motes numbered 5 and 22.

Figure 14(a) plots the temperature readings measured by these nodes throughout the course of the experiment. Figure 14(b) provides a better resolution of the temperature readings measured by the non-faulty nodes (all nodes other than 23). Two observations can be made from these figures: (1) The motes measure slightly different readings depending on their location, but they show almost the same trends of the increase and decrease in temperature, and (2) Mote 23 gets intermittently stuck at a considerably lower temperature value. Clearly, the corrupted measurement from the mote 23 can adversely affect the final average calculated at the sink, resulting into false positives and false negatives for a cold air drainage event.

Another fault which is not prominent in the above figures is the variable number of readings obtained from different motes. This can be attributed to a bad communication channel or to malfunctioning software/hardware components at

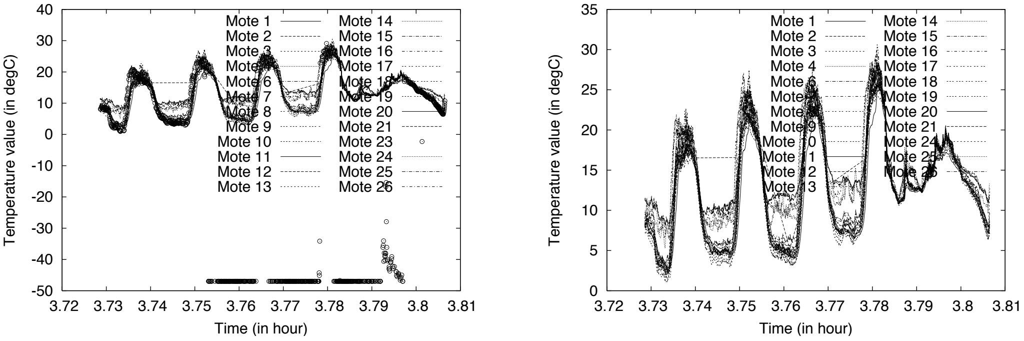


Fig. 14. Temperature measurements from the James Reserve sensor network deployment

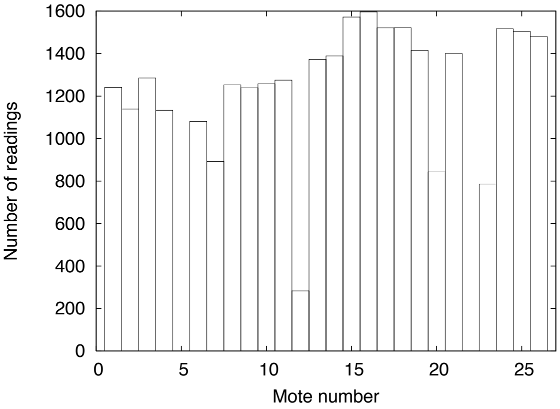


Fig. 15. Histogram of the temperature measurements taken by the nodes throughout the experiment

the corresponding nodes. Figure 5.2.1 plots the histogram of the total number of measurements collected at the base station corresponding to different nodes. It is hard to quantify the effects of this fault. If there are a large number of missed measurements, it can potentially result in a false positive. Furthermore, if the communication ambiguities are localized, the final average temperature calculated at the sink can be potentially many degrees off from the accurate value.

Our objective was to gauge if RFSN middleware can identify these faults. In order to achieve this, we replayed the same scenario as was witnessed by the motes in the real deployment in Avrora simulations. In addition, we ran the RFSN middleware service at the sink, so that it can establish reputation metrics for every node over the course of the experiment. For ease of analysis, we focus on just 10 motes for evaluation. These 10 motes were randomly selected; mote 1-4, 10, 16, 20, 23, 24 and 26. Figure 16(a) plots the trust metric that the base station establishes for different nodes. As can be observed from Figure 16(a), the trust metric of the faulty node, mote 23, goes down when it starts misbehaving. Note that mote 23 is not

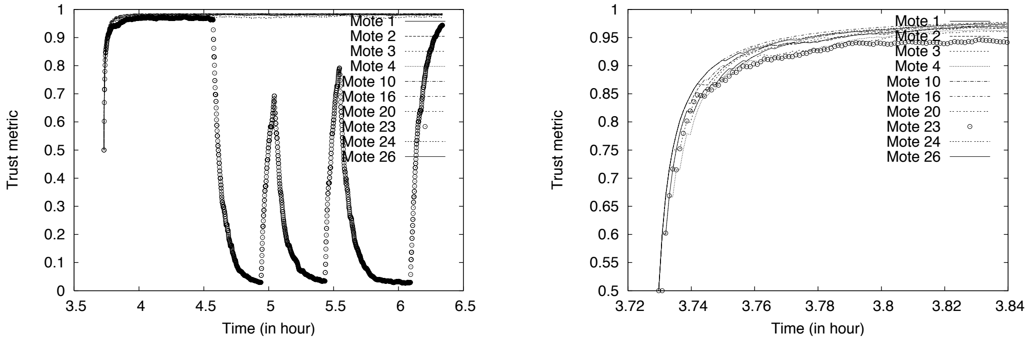


Fig. 16. Trust metric at the base station

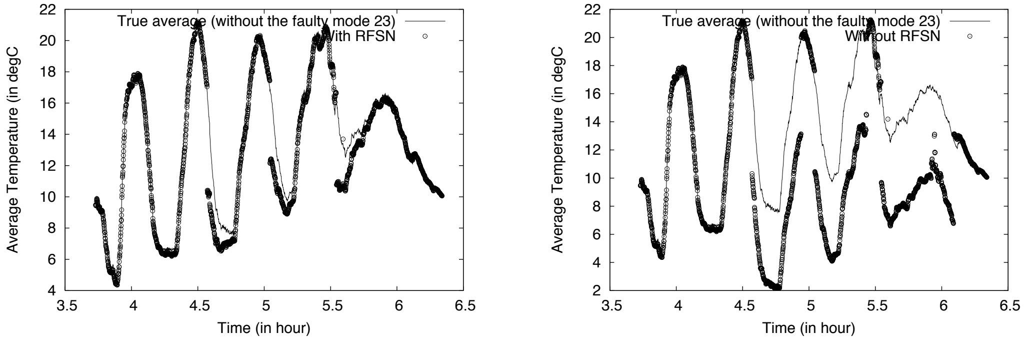


Fig. 17. Average temperature calculated at the base station

branded as a faulty node permanently. When the temperature reading of mote 23 is non-faulty, it does gain reputation.

In order to understand the impact of the variable number of readings from different nodes, we need to zoom in on the first few minutes of the experiment. Figure 16(b) plots the evaluation of trust for different nodes in the starting 20 minutes of the experiment. Note that all the motes, even the mote 23, are non-faulty during this duration. As a result, ideally every mote should have been able to establish the same trust metric at the base-station. However, each of the motes has a slightly different rate of the increase in the reputation. This is not an artifact of the different rating from the outlier detection scheme. All the motes are assigned exactly the same rating, as all of them are equally consistent. This is an artifact of missed measurements at the base-station. Note that mote 23 has the slowest rate as it also has the minimum number of readings (refer Figure 5.2.1). We note that eventually, over the complete course of the experiment, this disparity among the different motes goes away because of the large number of measurements.

Figure 17(a) and 17(b) plot the average temperature calculated at the sink, both with and without the system running RFSN, respectively. In both the cases, we compare the average temperature with the *true* value of the average temperature. We note that we do not have access to the *ground truth* of the average temperature

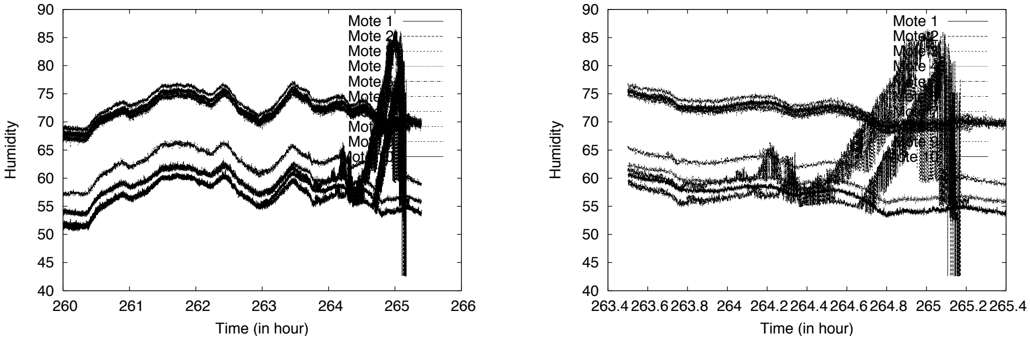


Fig. 18. Humidity data from an experiment with ten sensors inside a styrofoam box.

value; we calculate the average temperature over the rest of the non-faulty nodes and use it as a representative of the true value. As can be seen, the average calculated in the system using RFSN closely follows the true value. There are moments when there is huge disparity; this corresponds to the scenarios when the behavior of mote 23 suddenly changes. RFSN does take some time in converging to the accurate trust metric that can be associated with the mote 23. However, it eventually establishes the right metric and hence, the calculated average temperature closely tracks the true value.

5.2.2 *Humidity monitoring in a styrofoam box.* The second representative example comes from the deployment done in lab to find the variability in temperature and humidity sensors. In all the deployment consisted of 10 Mica2 motes, placed inside a styrofoam box, with MDA300 sensor boards to take the temperature and humidity measurements. As mentioned earlier in Section 2, one of the nodes had its temperature and humidity sensor shorted to one another, causing the temperature measurement to track the wrong phenomenon. It was easy for us to detect this fault using RFSN, and we do not cover it in detail in this paper due to space constraints. However, a more interesting behavior to observe is that of the humidity sensors.

The humidity sensor requires a 3.3V excitation voltage. The battery voltage is stepped up to this point for excitation. During the course of the experiments, the batteries drained to around 2.3-2.6V. As a result, some of the humidity sensors recorded inconsistent readings when the batteries were very low. Figure 18(a) shows the humidity readings measured at the 10 nodes over the course of the complete experiment. Figure 18(b) shows the zoom in version of the end of the experiment, wherein some of the humidity sensors started producing inconsistent readings.

This scenario is interesting because of several reasons - (1) The corrupted value from the humidity sensors are still within the range of the non-faulty values, and hence a simple consensus-based outlier detection on the humidity measurements will not be able to identify these misbehaving nodes, (2) There are multiple nodes, numbered 2, 3, 4 and 5, that go bad at some time during the experiment, and (3) Different motes have different faulty scenarios: Mote 4 is bad only for a very small duration, Mote 3 goes bad for a long period but it starts giving out good reading after that, Mote 5 goes bad for a small duration, then becomes good, again goes

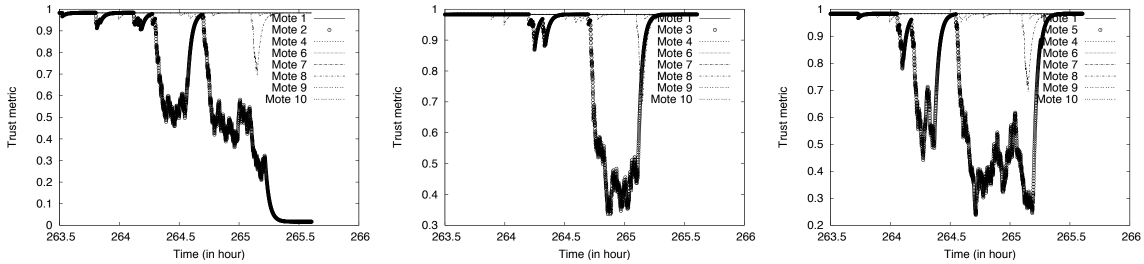


Fig. 19. Reputation for the ten sensors taking humidity measurements inside a styrofoam box.

bad for a long time and then starts giving out correct readings, Mote 2 shows the same behavior as mote 5 but it breaks down completely and stops giving out any readings towards the end of the experiment.

As shown in Figure 18(b), the absolute value of the humidity measurements cannot be designated as corrupted because it falls well within the range of the non-faulty values. It is only when we observe the change of these humidity measurements over time, we can identify the ongoing misbehavior in the network. Motivated by this, we decided to use the rate of the increase/decrease in humidity over time as the classifying metric for identifying misbehavior.

Figure 19 plots the reputation established at the sink using RFSN. Since the Watchdog mechanism (or the consensus-based outlier detection) works completely agnostic to the application context, it was trivial for us to use the rate of increase/decrease in humidity over time as the input to the Watchdog mechanism rather than the absolute humidity measurements. In order to provide a better visualization, we demonstrate the reputation using three graphs, one for each mote 2, 3 and 5 respectively. Note that the results were obtained using just one experimental run and not three separate runs. In fact, we wanted to gauge the performance of RFSN under the influence of multiple misbehaving nodes.

As can be observed from Figure 19, RFSN is able to identify the misbehavior of the four motes, 2, 3, 4 and 5. Note that the Figure 19 shows the reputation for the nodes towards the end of the experiment. The actual experiment was started at a time, equal to 260, much before the starting time scale of Figure 19, 263.5. Thereby, all the nodes have already been able to establish a good reputation, around 1.0, by the time scale reaches 263.5. Figure 19 clearly demonstrates the flexibility of RFSN. The trends of the reputation evolution for different misbehaving nodes are only dependent on their corresponding faulty behavior.

5.2.3 *Summary.* The experimental study in this subsection demonstrates the applicability of RFSN in identifying misbehavior in the context of real deployments. The sink, using the RFSN middleware service, is not only able to identify the misbehaving nodes in real time, it is also able to establish a relative magnitude of each node's misbehavior as compared to other misbehaving or good nodes. It can greatly help the end-user to reconfigure the system or take appropriate countermeasures, especially for short-term deployments such as Intel Research [int], CENS Bangladesh [Ramanathan et al.], when the real-time feedback about the operation of the network is paramount.

The sink, using the RFSN middleware service, is also able to fuse the data from multiple sensor nodes, simultaneously being resilient to corrupted readings from misbehaving or fault nodes. This can be especially useful in sense-response applications such as intruder detection, remote healthcare, home automation, etc., where actuation is performed based on the aggregated information from multiple nodes and a single corrupted reading can have huge life-critical or economic implications.

Although RFSN is lightweight and can be easily added to a sensor networking system, as shown in Section 4, it cannot function completely independently from the application context. The end-user does need to configure the RFSN middleware service appropriately. We saw an instantiation of this in our second representative example, wherein we had to change the input to the Watchdog from the absolute humidity measurements to the rate of increase/decrease of humidity in time. The modular architecture and the design of each building block of RFSN is free from any application specific context, and this allowed easy addition of a new outlier metric. All the application specific customization can be done through the APIs provided by the middleware service. Therefore, although we had to change the input to the Watchdog, it was a trivial change requiring no additional code.

In none of our experiments, we have used the concept of reputation integration and still we were able to identify faulty misbehavior successfully. This raises questions about its applicability to RFSN. Note that reputation integration step does not come at a free cost. If this is enabled, each node will be transmitting an extra packet (at a period, which can be configured by the application writer) containing its reputation metrics about other nodes. Having said so, the benefits of this step will be significant in dynamic topologies, wherein a node does not get enough chances to interact with other nodes to establish a meaningful reputation about them. Another domain, where reputation integration will be very useful is when RFSN is extended to counter malicious misbehavior, as discussed in the next section. Both these areas have not been covered in this paper and we leave a detailed analysis of the benefits vs cons of the reputation integration step for future work.

6. DISCUSSION

In this section, we will briefly discuss some of the related works that lie in the domain of resilient data aggregation and the applicability of RFSN to counter malicious attacks.

6.1 Resilient data aggregation

In [Wagner 2004], the authors propose a concrete mathematical framework for formally evaluating the security of several resilient aggregation techniques. The main focus of this work is to attract the attention of the researchers towards this area and make them realize the pitfalls of doing insecure data aggregation. Although no concrete solutions were proposed in the paper, a very simple and intuitive approach of throwing away the abnormal data was evaluated. In a way, this approach is equivalent to a single run of RFSN. We not only discard the bad data readings, but we integrate this information in reputation metrics which help us do efficient prediction in the future.

A related domain of work is of secure data aggregation. In [Du et al. 2003] and SIA [Przydatek et al. 2003], authors proposes a mechanism that allows the base station to check the aggregated values submitted by several designated aggregators, based on the endorsements provided by a certain number of witness nodes around the aggregators. In this model, the base station is the only aggregator. On the other hand, works such as [Zhu et al. 2004] and SDAP [Yang et al. 2006] focus on hop-by-hop secure data aggregation. The scheme in [Zhu et al. 2004] works only for a single compromised node, whereas SDAP is a more comprehensive solution, using the method of divide-and-conquer. We note that these works try to solve a different problem and cannot be simply extended for tolerating faulty node measurements. Note that a faulty measurement can come from a well authenticated node. We visualize RFSN working alongside one of these protocols to provide both resilient as well as secure data aggregation.

6.2 Malicious misbehavior

In this paper, we have focused on the applicability of RFSN to counter non-malicious faults resulting from corrupted data readings. The readings can also be deliberately corrupted if a few nodes in the network have been compromised. RFSN will be equally effective in countering such behavior, provided the number of compromised nodes are in a minority. This is mainly because RFSN acts on the misbehavior without figuring out the actual cause of the misbehavior.

Having said so, RFSN should not be envisioned as a scheme that can provide secure aggregation as well. First, RFSN does not provide any confidentiality or the authentication of the readings being reported by individual sensor nodes. Second, there is a big difference between malicious and non-malicious misbehavior. Faulty misbehavior is typically random and can be categorized in one of the classes as mentioned in the previous sections. As we have shown, RFSN can easily counter such misbehavior. On the other hand, RFSN will not be able to tolerate a much more planned attack that tries to abuse weaknesses in different building blocks of the framework. For example, a malicious attacker can modify the readings just enough so that they still pass the outlier detection. It can also try to abuse the reputation system or can potentially block the network traffic. We believe that RFSN functionality would need to be extended and perhaps supplemented with other schemes to provide a comprehensive solution against malicious misbehavior.

7. SUMMARY

Applications involving embedded sensing technology are slowly moving from academic contexts to wider scientific, industrial and social contexts. When used in life-critical or economic applications such as healthcare, entertainment, urban sensing, etc., a paramount challenge is to provide explicit data authentication mechanisms. This is because the misbehavior resulting from faulty nodes can drastically corrupt the information provided by the sensor network.

In this paper, we presented a generalized and unified approach for providing data authentication by modeling it as a problem of developing a community of trustworthy sensor nodes. We developed a Reputation-based Framework for Sensor Networks (RFSN), where each sensor node maintains reputation for other nodes. This reputation can be used as an inherent aspect in predicting the future behavior of the nodes, thereby allowing the identification of misbehaving nodes. RFSN integrates tools from statistics and decision theory into a comprehensive, distributed and completely scalable framework. We employ a Bayesian formulation, specifically a beta reputation system, for reputation representation, updates, integration and trust evolution. Consensus-based outlier detection schemes are used to associate a level of confidence with each data reading.

RFSN is available as a middleware service on Motes. The memory and energy overhead of using this service is negligible. It can be easily integrated in existing sensor networking deployments without making any modifications to the existing application software. The software design of RFSN is modular and the design of each building block os RFSN is free from any application specific context. This allows the end-user to easily customize the operation of RFSN to its desired application needs by appropriately configuring the RFSN parameters and making changes at the API level.

We evaluated the efficacy of RFSN using multiple contexts: (1) a lab-scale testbed of Mica2 motes, (2) Avrora simulations, and (3) real data sets collected from the sensor network deployment in James Reserve. RFSN is able to identify the misbehaving nodes for a variety of fault scenarios. Besides identifying the misbehavior of the nodes, we are also able to establish a relative magnitude of each node's misbehavior as compared to other misbehaving or good nodes. RFSN can be used in two ways. First, since RFSN provides a real time feedback about the node's misbehavior, it can greatly help the end-user to reconfigure the system or take appropriate countermeasures. Second, it can be used to aggregate information from multiple sensor nodes in a resilient manner, without getting affected by the corrupted readings generated by faulty nodes.

REFERENCES

James Reserve. http://www.jamesreserve.edu/.

Sensor Network deployment at Intel Research. htttp://www.intel-research.edu.

BALZANO, L. Taxanomy of faults in sensor networks. NESL Technical Report. htttp://nesl.ee.ucla.edu.

BARNETT, V. AND LEWIS, T. Outliers in Statistical Data. John Wiley, 1994.

BASAR, T. AND OLSDER, G. J. 1999. Dynamic noncooperative game theory. In *SIAM*.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTICS, A. The keynote trust management system. RFC 2704, 1999.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized trust managament. In *IEEE Conference on Security and Privacy*.

BREUNIG, M. M., KRIEGEL, H. P., NG, R. T., AND SANDER, J. 2000. LOF: Identifying density-based local outliers. In *Proceedings of the ACM SIGMOD Conference*. 93 –104.

BUCHEGGER, S. AND BOUDEC, J. L. 2002. Performance analysis of the CONFIDANT protocol. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*.

BUCHEGGER, S. AND BOUDEC, J. Y. L. 2003a. Coping with false accusations in misbehavior reputation systems for mobile in ad-hoc networks. In *EPFL Technical Report*.

BUCHEGGER, S. AND BOUDEC, J. Y. L. 2003b. The Effect of Rumor Spreading in Reputation Systems for Mobile Ad-hoc Networks. In *Proceedings of WiOpt Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*.

DELLAROCAS, C. 2000. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the second ACM Conference on Electronic Commerce*.

DU, W., DENG, J., HAN, Y. S., AND VARSHNEY, P. K. 2003. A witness-based approach for data fusion assurance in wireless sensor networks. In *Proceedings of IEEE Globecomm*.

Feng, J., Megerian, S., and Potkonjak, M. 2003. Model-based calibration for Sensor Networks. In *IEEE International Conference on Sensors*.

Ferguson, T. S. 1973. A bayesian analysis of some nonparametric problems. In *The Annals of Statistics*. Vol. 1. 209–230.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. 2003. *Bayesian Data Analysis*. Chapman and Hall.

Han, S., Kumar, R., Shea, R., Kohler, E., and Srivastava, M. B. 2005. A dynamic operating system for sensor nodes. In *Proceedings of the ACM Conference on Mobile Systems, Applications and Services (MobiSys)*. ACM Press.

Hill, J. and Culler, D. 2001. A wireless-embedded architecture for system level optimization. In *UC Berkeley Technical Report*.

Jew, R., Javelo, A., and Ganeriwal, S. Fault tolerant temperature monitoring system. NESL Technical Report. htttp://nesl.ee.ucla.edu.

Josang, A. 2001. A logic for uncertain probabilities. In *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*.

Josang, A. and Ismail, R. 2002. The beta reputation system. In *In Proceedings of the 15th Bled Electronic Commerce Conference*.

Knorr, E. M. and Ng, R. T. 1998. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *Proceedings of the 24th International Conference on Very Large Data Bases*. 392–403.

Knorr, E. M. and Ng, R. T. 1999. Finding Intensional Knowledge of Distance-based Outliers. In *Proceedings of the 25th International Conference on Very Large Data Bases*. 211–222.

Koushanfar, F. and Potkonjak, M. 2005. Markov-chain based models for missing and faulty data in mica2 sensor motes. In *IEEE International Conference on Sensors*.

Koushanfar, F., Potkonjak, M., and Sangiovanni-Vincentelli, A. 2002. Fault tolerance in wireless ad hoc sensor networks. In *IEEE International Conference on Sensors*.

Koushanfar, F., Potkonjak, M., and Sangiovanni-Vincentelli, A. 2003. On-line fault detection of sensor measurements. In *IEEE International Conference on Sensors*.

Koushanfar, F., Potkonjak, M., and Sangiovanni-Vincentelli, A. 2004. Error models for light sensors by non-parametric statistical snalysis of raw sensor measurements. In *IEEE International Conference on Sensors*.

Michiardi, P. and Molva, R. 2002. Core: A COllaborative REputation mechanism to enforce node cooperation in Mobile Ad Hoc Networks. In *Communication and Multimedia Security*.

Michiardi, P. and Molva, R. 2003. A game theoretical approach to evaluate cooperation enforcement mechanisms in mobile ad-hoc networks. In *Proceeding of WiOpt Modeling an Optimization in mobile, ad hoc and wireless networks*.

Papadimitriou, S., Kitawaga, H., Gibbons, P. B., and Faloutsos, C. 2002. LOCI: Fast Outlier Detection Using the Local Correlation Integral. In *Technical Report*.

Polastre, J., Hill, J., , and Culler, D. 2004. Versatile low power media access for wireless sensor networks. In *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys)*.

Przydatek, B., Song, D., and Perrig, A. 2003. SIA: secure information aggregation in sensor networks. In *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys)*.

Ramanathan, N., Balzano, L., Burt, M., Estrin, D., Harmon, T., Harvey, C., Jay, J., Kohler, E., Rothenberg, S., and M.Srivastava. Monitoring a Toxin in a Rural Rice Field with a Wireless Sensor Network. CENS Technical Report 62 2006. htttp://cens.ucla.edu.

Resnick, P., Kuwabara, K., Zeckhauser, R., and Friedman, E. 2000. Reputation systems: Facilitating trust in e-commerce systems. In *Communications of the ACM*.

Resnick, P. and Zeckhauser, R. 2000. Trust among strangers in Internet transactions: Empirical analysis of eBays Reputation System. In *Working paper for the NBER workshop on empirical studies of electronic commerce*.

Shafer, G. A mathematical theory of evidence. Princeton University, 1976.

SZEWCZYK, R., MAINWARING, A., POLASTRE, J., ANDERSON, J., AND CULLER, D. 2004. An analysis of a large scale habitat monitoring application. In *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys)*.

TITZER, B., LEE, D., AND PALSBERG, J. 2005. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of IPSN Track on Sensor Platform, Tools and Design Methods for Networked Embedded Systems*.

WAGNER, D. 2004. Resilient aggregation in sensor networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*.

XIONG, L. AND LIU, L. 2003. A reputation-based trust model for peer-to-peer ecommerce communities. In *IEEE Conference on e-commerce*.

YANG, Y., WANG, X., ZHU, S., AND CAO, G. 2006. SDAP: a secure hop-by-hop data aggregation protocol for sensor networks. In *Proceedings of International Symposium on Mobile Ad Hoc Networking and Computing*.

ZHU, S., SETIA, S., JAJODIA, S., AND NING, P. 2004. An integrated hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *Proceedings of IEEE Symposium on Security and Privacy*.