Online Outlier Detection in Sensor Data Using Non-Parametric Models

S. Subramaniam
UC Riverside
sharmi@cs.ucr.edu

T. Palpanas
IBM Research
themis@us.ibm.com

D. Papadopoulos, V. Kalogeraki, D. Gunopulos
University of California, Riverside
dimitris,vana,dg@cs.ucr.edu

ABSTRACT

Sensor networks have recently found many popular applications in a number of different settings. Sensors at different locations can generate streaming data, which can be analyzed in real-time to identify events of interest. In this paper, we propose a framework that computes in a distributed fashion an approximation of multi-dimensional data distributions in order to enable complex applications in resource-constrained sensor networks.

We motivate our technique in the context of the problem of outlier detection. We demonstrate how our framework can be extended in order to identify either *distance-* or *density*-based outliers in a single pass over the data, and with limited memory requirements. Experiments with synthetic and real data show that our method is efficient and accurate, and compares favorably to other proposed techniques. We also demonstrate the applicability of our technique to other related problems in sensor networks.

1. INTRODUCTION

Advances in processor technologies and wireless communications have enabled the deployment of small, low cost and power efficient sensor nodes in both civil and military settings [43, 24]. In such settings, an important consideration is how to monitor the physical environment and highlight the events of interest. In fact, environmental monitoring is one of the earliest applications of sensor networks. The context of the sensor networks makes these problems more challenging: First, sensors have *limited* resource capabilities, and second, data coming from many different streams may need to be examined *dynamically*, and *combined* to solve the problem at hand. In such an environment, it is important to process as much of the data as possible in a decentralized fashion, so as to avoid unnecessary communication and computation costs. Another important goal is that the entire process has to be self-managed, so that it can work in unattended environments over extended periods of time and automatically adapt to the changes in the environment of the sensor network.

VLDB '06, September 12-15, 2006, Seoul, Korea.

Recently proposed techniques [16, 20], that operate on approximations of the sensor data distributions, show that this general approach achieves substantial energy savings for the network. The approximate mode of processing is extremely useful for two reasons. First, it allows us to answer queries fast and cheaply, since we do not need to visit all the nodes of the network relevant to the query in order to get the answer. Second, it enables the execution of queries that we would otherwise not be able to answer without consuming a lot more resources. These are queries for which the computation of the exact answer requires many more resources than what are available in a sensor network. An example of such a query is the identification of outliers, or deviations[1] [28].

In this work, we propose a general and flexible data distribution approximation framework that does not require a priori knowledge about the input distribution. Then, based on this framework, we describe an efficient technique for distributed deviation detection in a sensor network. The goal is to identify, among all the sensor readings in a sliding window, those values that have very few near neighbors. Note that this is a challenging problem, even for static datasets [28, 38, 10, 36]. This problem is especially important in the sensor network setting because it can be used to identify faulty sensors, and to filter spurious reports from different sensors. Even if we are certain of the quality of measurements reported by the sensors, the identification of outliers provides an efficient way to focus on the interesting events in the sensor network.

To further motivate our work, we discuss the following example that we believe has significant practical benefits from our approach. Assume a machine that is fitted with sensors that monitor its operation. These sensors measure quantities such as temperature, pressure, and vibration amplitude for the different parts of the machine. If there is some malfunction or any other abnormality, some of these readings will deviate significantly from the norm. Note that such deviations may occur at different levels. For example, it may be the case that a small part of the engine is overheated when compared to the rest of the engine, or that the entire engine is overheated when compared to the rest of the machine. Finally, in some cases we have to monitor two specific attributes together, such as operating frequency and vibration amplitude, or otherwise we would miss interesting deviations. Therefore, we would like to be able to identify *multi*-dimensional outliers. The engine system will greatly benefit from techniques that can quickly identify such deviations and allow the engineers to correct them during operation.

[1]For the rest of this paper we will use the terms *outlier* and *deviation* interchangeably.

In this work we develop efficient online techniques for estimating the data distribution of sensor readings, for a sliding window. Our approach uses *kernel density estimators* [40] to approximate the sensor data distribution. Once we have estimated the data distributions, we can compute the density of the data space around each value, and therefore determine which values are outliers. Our experimental results show that the above technique is appropriate for the sensor network context: it is effective, has relatively low resource requirements, and its operation does not require any parameter settings. The modifications we propose allow our technique to operate efficiently in an online fashion. We also demonstrate how the necessary computations can be distributed in the sensor network, and organized in a hierarchy to achieve scalability. Below we summarize our main contributions:

- We propose online techniques for outlier detection in a sensor network. Our techniques are based on the efficient, in-network approximation of the input data distributions (i.e., the sensor data), and can effectively extend to more than one dimensions. These approximations can also serve for other applications, such as online estimation of range queries.
- We demonstrate the versatility of our framework by describing how it facilitates the implementation of two different methods for identifying outliers. The first uses a fast, distance-based algorithm [28], while the second employs a more robust, local metrics based approach [36].
- Our techniques operate efficiently in an online fashion, and their operation can be distributed across the nodes of the sensor network, efficiently utilizing a hierarchical organization of the network, and making the approach scalable to a large number of sensors.
- Finally, we describe the prototype implementation of our system, and through an extensive set of experiments using both synthetic and real datasets we demonstrate that our approach is efficient and accurate.

2. SENSOR NETWORK MODEL

We begin with a brief introduction of the sensor network model. We assume an event-based sensor network that consists of a set of sensors (each having a location on a 2-d plane) used to monitor and report observed events. When we deal with very large sensor networks, we have to take into account the issue of scalability of the query processing technique with respect to the size of the network. To that effect, we adopt a hierarchical organization for the sensor network, similar to the one used in [46]. The idea is to organize the network using overlapping virtual grids. We define several tiers for the grid with different levels of granularity, ranging from small local areas at the lowest tier, to the entire network area at the highest tier (see Figure 1).

At each cell at the lowest tier of the grid, there is one leader (or *parent*) node, that is responsible for processing the measurements of all the sensors in the cell. Moving up the hierarchy, the leader node of a cell collects values from the leader nodes of all its sub-cells in the lower level. For example, in Figure 1, node D_2 corresponds to a leader node in the third level of the hierarchy and processes the measurements from leader nodes A_1, B_4, C_5 and D_5.

The hierarchical decomposition of the sensor network, as well as the selection of the leaders for each level of the hierarchy, can be achieved using any of the techniques proposed in the literature [17, 33, 47]. These techniques ensure the leadership role is rotated among the nodes of the network, and describe protocols that achieve this in an energy efficient manner.

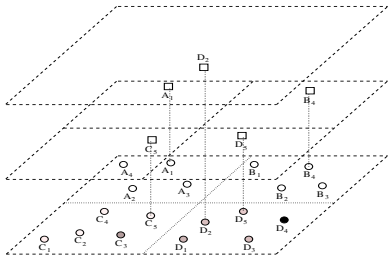


Figure 1: Hierarchical organization of a sensor network.

3. DISCOVERING ABNORMAL BEHAVIOR

There exist several formal definitions of an outlier. In our work, we follow two of the commonly-used definitions:

Distance-based outliers [28]: A point $\mathbf{p}$ in a dataset T is a (D, r)-outlier if *at most* D of the points in T lie within distance r from $\mathbf{p}$. The approach to detect such outliers does not require any prior knowledge of the underlying data distribution, but rather uses the intuitive explanation that an outlier is an observation that is sufficiently far from most other observations in the dataset. The usefulness and validity of distance-based outliers has been established in the literature [6, 28, 38].

Local metrics-based outliers [36]: This method detects outliers based on the metric Multi Granularity Deviation Factor (MDEF). For any given value $\mathbf{p}$, MDEF is a measure of how the neighborhood count of $\mathbf{p}$ (in its *counting* neighborhood) compares with that of the values in its *sampling* neighborhood. A value is flagged as outlier, if its MDEF is (statistically) significantly different from that of the local averages. The parameters r, the *sampling* neighborhood and αr, the *counting* neighborhood, determine the range over which the neighborhood counts are estimated. This method takes into consideration the local density variations in the feature space, and provides an automatic cut-off for the outliers, based on the characteristics of the data. Similar approaches and their applications can be found in [10, 39].

Distance based-outliers are very useful if the user is confident that the threshold can be specified accurately. If this is the case, the definition of distance-based outliers allows very efficient outlier detection. However, there are situations where the above definition of outliers may not be applicable. For example, if the data points exhibit different densities in different regions of the data space or across time, then using a single threshold value for identifying outliers may not be appropriate. Therefore, we have to employ more robust statistical techniques that compare the relative differences between observed points. In such cases, the use of MDEF for outlier detection can be expected to offer better results.

For the purposes of this study, we focus on identifying outliers in a data stream S, which represents a series of data points drawn from an unknown d-dimensional data distribution. More specifically, we are interested in finding the outlying values within a sliding window W that holds the last $|W|$ (d-dimensional) values of S. We are also interested in reporting outlying values in the union of the readings coming from multiple sensors. This is an important process, because it allows us to identify interesting values at different levels of

granularity and in larger network areas. Assume a number l of nodes, $n_1, \ldots, n_l$, called children, that read values from l different streams, $S_1, \ldots, S_l$. Let another node, n_p, be the parent of those l nodes. Then, n_p sees the combined values from all l individual sliding windows, and we would like to identify deviations in this set of values.

Note that in every case the goal is to identify outlying values in a sliding window, with respect to the rest of the data stream values in the window.

EXAMPLE 1. *Refer to Figure 1, where the sensor nodes reside in the bottom level (the other two tiers are conceptual). The sensors at the bottom level generate measurements, and they maintain a sliding window over the most recent measurements. At that level, each sensor is identifying outliers in its own sliding window (the outliers considered in this example are distance-based outliers). Let C_5 identify some outliers with moderately high values (denoted by the darker color of the node). Let D_4 identify outliers with the highest values among all the outliers identified by any sensor in this level.*

When we move up in the hierarchy, the aim is for each leader node to detect outliers within a sliding window that encompasses the measurements of all *the sensors that belong to its cell. Thus, node C_5 in the second level of the hierarchy identifies outliers with respect to the measurements of all the sensors in the* front left *quarter of the network. The best candidates for being marked as outliers by C_5 are the high values coming from sensor C_3, since these values are different from the values of all the other sensors in the same quarter of the grid. When we examine leader node D_5, the same is true for the high values coming from sensor D_4.*

In the third level of the hierarchy, the leader node D_2 is responsible for detecting outliers with respect to the values of all the sensors in the network. In this case, the values that are most probable for being outliers are the high values coming from sensor D_4, since no other sensor in the network reports values in that range. The same is not true for the values of sensor C_3, because the sensors D_1, D_2, D_3, and D_5 have similar readings.

As the previous example shows, we can choose to identify outliers at any level of detail, as defined by the hierarchy of the sensor network, and at various regions of the network. We stress that our framework provides the flexibility of identifying outliers at different levels of the hierarchy. The analyst has to decide the extent to which she will exploit this feature.

4. ESTIMATING THE DATA DISTRIBUTION

In this section we present a general framework for estimating the underlying distribution of the sensor readings in a sliding time window W. We note here that this problem is more general than simply finding outliers among the values of a given sensor, or even among many sensors in a given region. However, as we show in the next section, the problem of finding outliers can be solved efficiently if an accurate approximation of the data distribution can be found. In addition, the use of a data distribution approximation allows us to combine the information from many sensors efficiently, thus minimizing the communication costs required to find outliers among the values of different sensors. Finally, we also show that if we are able to compute an approximation of the data distribution efficiently in-network, other queries can be computed in the network as well, including finding spatial regions where a given property holds.

Estimating the Probability Density Function: There are several model estimation techniques that have been proposed in the literature, such as histograms [25, 21], wavelets [12, 18], kernel density estimators [40], and others. In our framework, we choose to estimate the distribution of the values generated by the sensors using the kernel density estimators, because of the following desirable properties: (i) they are efficient to compute and maintain in a streaming environment, (ii) they can very effectively approximate an unknown data distribution, (iii) they can easily be combined and (iv) they scale well in multiple dimensions. In general, it is computationally more expensive to apply the above operations in histograms or wavelets. Even though *sketches* can be used to approximate histograms and wavelets in an online setting [18, 42, 13], previous studies have also shown that kernels are as accurate as those two techniques [23, 8]. In Section 10, we compare the results obtained using kernel density estimators to histograms.

Kernel Estimators: The simplest statistical estimator for estimating the probability density function is random sampling. The *kernel estimator* [40, 8] is a generalized form of sampling, whose basic step is to produce a uniform random sample. As in random sampling, each sample point has a weight of one. In kernel estimation however, each point distributes its weight in the space around it. A *kernel function* describes the form of this weight distribution, generally distributing most of the weight in the area near the point. Summing up all the kernel functions we obtain a density function for the dataset.

More formally, assume that we have a static relation, T, that stores the d-dimensional values $\mathbf{t}$, $\mathbf{t} = (t_1, \ldots, t_d)$, whose distribution we want to approximate. The recorded values must fall in the interval $[0, 1]^d$. This requirement is not restrictive, since we can map the domain of the input values to the interval $[0, 1]^d$. Let R be a random sample of T, and $k(\mathbf{x})$ a d-dimensional function of $\mathbf{x} = (x_1, \ldots, x_d)$, such that $\int_{[0,1]} k(\mathbf{x})d\mathbf{x} = 1$, for all tuples in R. We call $k(\mathbf{x})$ the *kernel function*. We can now approximate the underlying distribution $f(\mathbf{x})$, according to which the values in T were generated, using the following function

$$f(\mathbf{x}) = \frac{1}{|T|} \sum_{\mathbf{t_i} \in R} k(x_1 - t_{i_1}, \ldots, x_d - t_{i_d}). \quad (1)$$

The choice of the kernel function is not significant for the results of the approximation [40]. Hence, we choose the Epanechnikov kernel that is easy to integrate:

$$k(\mathbf{x}) = \begin{cases} \left(\frac{3}{4}\right)^d \frac{1}{B_1 \ldots B_d} \prod_{1 \le i \le d} \left(1 - \left(\frac{x_i}{B_i}\right)^2\right) \\ \qquad \text{, if } \forall i,\ 1 \le i \le d,\ \left|\frac{x_i}{B_i}\right| < 1 \end{cases} \quad (2)$$

where $\mathbf{B} = (B_1, \ldots, B_d)$ is the bandwidth of the kernel function. We use Scott's rule to set $\mathbf{B}$ [40, 8]: $B_i = \sqrt{5}\sigma_i |R|^{-\frac{1}{d+4}}$, where σ_i is the standard deviation of the values in T in dimension i.

5. ONLINE APPROXIMATION OF THE DATA DISTRIBUTION IN A SLIDING WINDOW

In the sensor network setting we require that each sensor maintains a model for the distribution of values it generates. Since we are not interested in the entire history of the values produced by the sensors, it suffices to consider the values in a sliding window W of size N. Then, T holds only the values in this sliding window, i.e., the N most recent values.

At each point in time, we are interested in approximating the distribution of the data values within the sliding window. This procedure is illustrated in Figure 2, for two time

instances (2-d data). Figure 2(a) shows the sliding window and the distribution of the corresponding data. As time advances (Figure 2(b)), a different set of points falls inside the sliding window. Our aim is to use the kernel estimators for computing an approximation of the new data distribution at each point in time. The first step for creating a kernel estimator for a sliding window W is to maintain online a random sample of the set T that contains the values in the most recent window W. The other quantity we need for the kernel estimator is the standard deviation σ of the values in the sliding window W. Both of these operations can be efficiently supported in a data streaming environment.

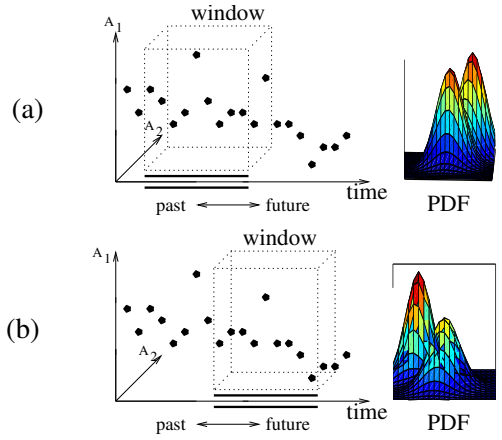


Figure 2: Estimation of the data distribution in the sliding window for two time instances (2-d data).

We use the "chain-sample" algorithm for producing a uniform random sample of size $|R|$, of a sliding window. The algorithm [4] starts with an initial random sample, and proceeds as follows. For each point in the sample, it picks at random the next element from the data stream that will replace it. The only restriction is that the new value must replace the old one, before the old one expires from the sliding window (that is, they should not be more than N values apart in the stream). The memory requirements are $O(d|R|)$.

The estimate for the standard deviation of the sliding window is computed using a concise histogram along the time axis [5]. The estimate of the standard deviation is derived by combining the statistical information stored in all the buckets of the histogram. The required memory is $O(\frac{d}{\epsilon^2}\log|W|)$, where ϵ is the maximum relative error we wish to tolerate in the estimation, and $|W|$ is the size of the sliding window.

Both the above algorithms are effective, computationally efficient, and have a very small memory footprint. We should also note that the above techniques allow our estimation method to effectively model distributions that change over time. This is true, because both the sample and the bandwidth, are incrementally recomputed at every time step.

5.1 Distributed Computation of Estimators

Combining Multiple Estimator Models: Assume we have a sensor network similar to the one shown in Figure 1. For simplicity, we use an example of a parent node with only two children nodes. The necessary extensions to multiple children are straight-forward. In order to identify global outliers, we need to build a model in the leader node that describes the combined data distribution of its two children. Each sensor node in the hierarchy can compute its own sample of the values that are observed by the sensors that belong in the subtree rooted at itself. Assume for example that a parent has two children that are leaves (sensors) and observe values at the same rate. The parent decides on an arbitrary order between the children. This creates a sequential ordering of the values observed by the two sensors, that interleaves the values. Then the parent can apply the "chain-sample" algorithm to sample the values of the two children in a time window. This scheme however is not flexible, as all sensors must be active and observe values at a fixed rate. It is also inefficient: although the actual sampling must be done at the leaves, each sensor must sample separately for each node on the path to the root. Since the samples of these nodes are different, each of the sampled values contributes only to one of the samples along the path to the root.

In our framework, we propose a more efficient mechanism for model composition. This allows us to take the data distribution models of two different sensors in the network and construct a single model that describes the behavior of the data of both sensors. This combined model is the model of a sensor at a higher level of the hierarchy.

In the case of kernel density estimators we have to combine the sample set, R, and the bandwidth of the kernel function, B (refer to Section 4). We combine the sample sets by taking their union. We may reduce the size of the resulting set by sampling from the two sets, using the techniques of [11]. In order to combine the bandwidths of two kernel functions, we only need to combine the standard deviations upon which the bandwidths depend. This is accomplished using the same techniques as the ones for computing the standard deviation in a sliding window of streaming data [5]. More specifically, we use the formula

$$V_{1,2} = V_1 + V_2 + \frac{N_1 N_2}{N_{1,2}}(\mu_1 - \mu_2)^2, \qquad (3)$$

where V is the variance (multiplied by N), $N_{1,2} = N_1 + N_2$, and $\mu_{1,2} = \frac{\mu_1 N_1 + \mu_2 N_2}{N_{1,2}}$.

The above process gives to the high level sensors a coarse view of the sensor network, where the details specific to different parts of the deployment area have been masked away. If we wish to examine the individual values of the sensors we have to query the low level sensors.

Propagating Estimator Updates in the Network Hierarchy: An interesting question is how often a node should send its model to the leader of the cell it belongs to. Obviously, the answer depends on the data distribution of the node, and how fast it is changing.

As we discussed earlier, the kernel density estimators can successfully approximate a distribution that changes over time, because they adapt their parameters (i.e., sample and bandwidth) with every new value that comes in. However, the same is not true for the leader nodes in each cell of the hierarchy. The leader nodes do not see the sensor data values, but rather rely on their children to inform them of any changes in the input data distributions that they should incorporate in their own models.

The simplest approach is to have the children transmit updates to the parent as these updates take place in their own estimators. Assume that the parent has l children, each having a kernel estimator of size $|R|$, and that the kernel estimator of the parent has size $|R_p|$. Then, with probability $f = \frac{|R_p|}{l|R|}$, when a child updates its kernel estimator by adding a new kernel, it also propagates this update to its parent (i.e., it transmits the new kernel and the new standard de-

viation(s)). This simple approach can be used to efficiently maintain a sample with expected size $|R_p|$ at the parent, and this is what we use in our implementation.

The above scheme has the important advantage that the parent's distribution quickly adjusts to changes in the distribution of the observed data, and as we show in the experiments can be implemented efficiently, with a relatively small number of messages. However, we can fine-tune this technique if we allow each sensor to monitor the distribution of the data it observes. When there are small changes, a sensor can reduce f, in effect reducing the update rate. When large changes are observed, the sensor can set f close to 1, thus essentially propagating its entire kernel estimator. Detecting those distribution changes is a difficult task. We can use elaborate techniques that are specifically designed to identify changes in (unknown) distributions of streaming data [7], or simple approaches, like monitoring the first moments of the data distribution (i.e., mean, standard deviation, and skew).

5.2 Fault Tolerance

In our framework, fault tolerance can be achieved in a straight-forward manner, with no need for special provisions. When a leader node fails, we simply have to elect a new node [33] to act as the leader for the particular cell in the hierarchical grid. The new leader can recover the full state of the old leader, by combining the models of its children, and simply has to start consuming the information provided by its children, from the point that the old leader left off. Note that the above sequence of actions does not result in the loss of any information.

5.3 Complexity Analysis

In this section, we analyze the storage complexity of the kernel model at each sensor, and the time complexity for processing a range query with a kernel estimator model. Since sensors have limited memory and processing capabilities, it is very important for us to ensure that the estimator model has a small memory and computational requirements.

As described in Section 5, the sensors have to store the following: sample of the current window of the input data stream, which requires $O(d|R|)$ memory, and some extra storage for the computation of the standard deviation of the values in the sliding window, which is $O(\frac{d}{\epsilon^2}\log|W|)$. The following theorem summarizes the memory requirement.

THEOREM 1. *The memory requirement for a sensor to keep an estimate of its distribution is $O(d(|R|+\frac{1}{\epsilon^2}\log|W|))$, where d is the data dimensionality, $|R|$ is the size of the sample, ϵ is the maximum error for the estimation of the standard deviation, and $|W|$ is the size of the sliding window.*

THEOREM 2. *Assuming the Epanechnikov kernel function, the time complexity to answer a range query $N(\mathbf{p}, r)$ with the kernel estimator model is $O(d|R|)$, where d is the data dimensionality, $|R|$ is the sample size, and r is the range specified by the query.*

PROOF. The number of measurements in the current window that are within a range r from a value $\mathbf{p}$ is given by

$$N(\mathbf{p}, r) = P[\mathbf{p} - r, \mathbf{p} + r] \times |W|. \quad (4)$$

Let us denote the probability $P[\mathbf{p}-r, \mathbf{p}+r]$ as $P(\mathbf{p}, r)$. Thus, from Equation 1

$$P(\mathbf{p}, r) = \frac{1}{|R|} \int_{[\mathbf{p}-r,\mathbf{p}+r]} \sum_{\mathbf{t_i} \in R} k(\mathbf{x} - \mathbf{t_i}) d\mathbf{x}. \quad (5)$$

In order to estimate this, we have to access those points in the random sample R that have a non-zero contribution to the query range $(\mathbf{p} - r, \mathbf{p} + r)$, and compute the integral of their Epanechnikov function.

From the definition of the Epanechnikov kernel in Equation 2, we see that the measurements in the random sample which have non-zero contribution are in the range $(p_i - r - B_i, p_i + r + B_i)$ for all dimensions i. Let $R' \subseteq R$ represent all the above measurements. Also, the Epanechnikov kernel is a quadratic function and its integral has a closed form. Thus, the computation of the integral for one measurement requires *constant time*. Then, the range query requires a search time of $O(d|R|)$, an additional time of $O(d|R'|)$ to to compute the integral of the measurements in R'.

In the 1-dimensional case ($d = 1$), we can answer the range query in time $O(\log|R| + |R'|)$, by simply maintaining the sampled values in sorted order (and using binary search). □

6. COMPARING DISTRIBUTIONS

We now discuss a method for computing the difference between two distributions, which will be useful for the algorithms we describe. Several methods have been proposed to quantify the difference between probability density distributions [29]. One widely used measure is the *Kullback-Liebler divergence* $D(p \parallel q)$ [14], which is defined as

$$D(p \parallel q) = \int_{\mathbf{y}} p(\mathbf{y})(\log p(\mathbf{y}) - \log q(\mathbf{y})), \quad (6)$$

where $p(\mathbf{y})$ and $q(\mathbf{y})$ are probability distribution functions over $\mathbf{y}$, and $\mathbf{y}$ is drawn from a finite set $\mathbf{Y}$. However, the measure is undefined when $p(\mathbf{y}) > 0$ but $q(\mathbf{y}) = 0$ for some $\mathbf{y} \in \mathbf{Y}$. The $KL-divergence$ is therefore not applicable to the density distributions derived by kernel density estimation method, because this method may assign probability of zero for regions in the domain of the values. We use a variation of the KL-divergence, called the *Jensen-Shannon divergence* [30, 22] which is defined as follows

$$JS(p, q) = \frac{1}{2}\left[D(p \parallel avg(p, q)) + D(q \parallel avg(p, q))\right] \quad (7)$$

where $avg(p, q)$ is the average distribution $(p(\mathbf{y}) + q(\mathbf{y}))/2$.

We estimate the JS-distance between two kernel estimator models $p(\mathbf{x})$ and $q(\mathbf{x})$ as follows. We approximate the estimated distribution with the values of the function with a finite set of grid points $\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_k}$. Thus, we approximate the term $D(p \parallel avg(p, q))$ in Equation 7 as

$$D(p \parallel avg(p, q)) = \sum_{i=1\ldots k} P_p(\mathbf{b_i}, bs/2) \times \left[\log(P_p(\mathbf{b_i}, bs/2)) - \log(\tfrac{P_p(\mathbf{b_i}, bs/2) + P_q(\mathbf{b_i}, bs/2)}{2})\right] \quad (8)$$

where bs is the grid interval and P_p and P_q are the probabilities estimated with respect to the estimator models $p(\mathbf{x})$ and $q(\mathbf{x})$ respectively. We approximate the term $D(q \parallel avg(p, q))$ in Equation 7 in a similar way, and estimate the JS-divergence between the kernel estimator models. The time complexity for the above procedure is $O(dk|R|)$.

7. DISTRIBUTED DETECTION OF DISTANCE-BASED OUTLIERS

In this section, we describe our technique for detecting distance based outliers in sensor networks, in a distributed manner. Given a new observation $\mathbf{p}$, the sensor can use its current

density distribution function $f(\mathbf{x})$, to estimate the number of values that are in the neighborhood of $\mathbf{p}$. This allows us to identify distance-based outliers [28]. Specifically, we estimate the number of values, $N(\mathbf{p}, r)$, in T that fall in the interval $[\mathbf{p}-r, \mathbf{p}+r]$, using Equation 4. If this number is less than an application-specific threshold t then $\mathbf{p}$ is flagged as an outlier.

We now turn our attention to the task of identifying outliers in the parent node. Remember that a parent node combines in a single pool all the data that its children process. Consequently, outliers are identified with respect to this new pool of data. One might think that in order to successfully report the outliers the parent node has to read in all the data from its children's input data streams, and for each data value determine whether it is an outlier or not. Fortunately, this is not true. It suffices for the parent node to examine only the values that have been marked as outliers by its children. All the other data values can be safely ignored, since they cannot possibly be outliers.

THEOREM 3. *Assume nodes $n_1, \ldots, n_l$ children of node n_p, data streams $S_1, \ldots, S_l$ referring to the l children nodes, and corresponding sliding windows $W_1, \ldots, W_l$. The sliding window of node n_p is defined as $W_p = \bigcup_{i=1}^{l} W_i$. Let, at some point in time, $O_1, \ldots, O_l$ be the sets of distance-based outliers corresponding to the l sliding windows. Then, for the set O_p of outliers in W_p it holds that $O_p \subseteq \bigcup_{i=1}^{l} O_i$.*

This theorem is important for two reasons. First, it results in significant computation savings for the parent node, because it only needs to examine a very small subset of the streaming values, i.e., the outliers identified by its children. Second, it limits the necessary communication messages from the children nodes to their parents. Both are desirable properties in a sensor network.

For detecting distance based outliers, we propose the *D3 (Distributed Deviation Detection)* algorithm. For the presentation of the algorithm, we refer to all the leader nodes as *parent* nodes, and to the rest as *leaf* nodes. The *D3* algorithm (Figure 4) starts by initializing the *LeafProcess* procedure (lines 11-20) in each leaf node. These nodes compute and maintain a kernel density estimation model, for approximating the input data distribution. Based on this model, they report any values that satisfy the criterion for being an outlier. Each time an outlying value is identified, it is transmitted to the corresponding parent node and is checked against the model of that parent node, in order to determine whether this value is also an outlier in that level of the hierarchy. Along with the identified outliers, a sensor node may also send to its parent its current sample (depending on f).

The time complexity of the *D3* algorithm is dominated by the *IsOutlier()* procedure (Figure 4, lines 32-36), the procedure that checks if a value $\mathbf{p}$ is an outlier by computing $N(\mathbf{p}, r)$. As described in Section 5.3, the computation of $N(\mathbf{p}, r)$ takes $O(d|R|)$ ($O(\log|R| + |R'|)$ if $d = 1$) and the memory requirement for each of the sensors is $O(d(|R| + \frac{1}{\epsilon^2}\log|W|))$. Even if we set the parameters to "large" values, that is, 20000 for $|W|$, 2000 for $|R|$, and 0.2 for ϵ, the total memory usage for each sensor is less than 10KB. The resource requirements of the approach we propose are well within the capabilities of the state of the art sensors. There currently exist sensors (e.g., *Intel Mote* [27], and *MICA2DOT* [1]) in less than half the size of a matchbox, that run on 12MHz processors, have more than 512KB of memory, and achieve communication throughput rates in excess of 75KB per second.

8. OUTLIER DETECTION USING MULTI-GRANULAR LOCAL METRICS

We will now examine how local metrics-based outliers can be detected in a sensor system. For a new observation $\mathbf{p}$, a sensor can use its density estimator model to determine if $\mathbf{p}$ is an MDEF based outlier with respect to its data stream.

From the definition of the MDEF-based outliers (Section 3), we can observe that they are non-decomposable. That is, an observation detected as outlier in a parent sensor *need not* have been an outlier in its children sensors. Therefore, Theorem 3 is not true for MDEF-based outliers and we can not follow a technique similar to that of *D3* to detect outliers in parent sensors. Due to this reason, only the leaf sensors detect outliers. A leaf sensor can report outliers with respect to the rest of the values it is observing, as well as with respect to the values observed in an entire region in which it belongs. This is achieved by having a leader node in the higher levels of the hierarchy communicate its probability density function estimate, which we will refer to as the *global* model, to the leaf sensor.

In order to detect MDEF-based outliers in sensor networks, we propose the *MGDD (Multi Granular Deviation Detection)* algorithm (Figure 4). The sensor nodes at the lowest level have a copy of the *global* probability density function, in addition to the *local* estimation model of their input data stream. The MDEF-based outliers are estimated based on the *global* estimation model. In Section 8.1, we discuss how the *global* estimation model is updated in each of the leaf sensors.

The *isMDEFOutlier()* function mentioned in line 27 in Figure 4 is the *aLOCI* algorithm in [36], and hence we briefly describe it here. To estimate if the observation is an outlier, two values are required to be computed based on the density model (a) the *αr-neighbors* of the observation $\mathbf{p}$ and (b) the number of observations in each of the $2\alpha r$ interval of the domain. We illustrate in Figure 3, how the above values are computed from a kernel estimator model (for 1-d). From these, we compute $MDEF(\mathbf{p}, r, \alpha)$ i.e, the deviation factor of the observation $\mathbf{p}$ and $\sigma_{MDEF}(\mathbf{p}, r, \alpha)$ i.e., the normalized standard deviation in the sampling neighborhood of $\mathbf{p}$. The new observation $\mathbf{p}$ is flagged as an outlier if

$$MDEF(\mathbf{p}, r, \alpha) > k_\sigma \sigma_{MDEF}(\mathbf{p}, r, \alpha), \tag{9}$$

where k_σ is the factor which determines what is a significant deviation.

THEOREM 4. *The computational complexity to detect if a new observation is an MDEF-based outlier is $O(\frac{d|R|}{2\alpha r})$ and the memory usage is $O(d|R|)$, where r is the* sampling *neighborhood and αr is the* counting *neighborhood.*

PROOF. The sensors have to maintain a copy of the *global* estimator model, in addition to the local estimator model, requiring $O(d|R|)$ memory. The algorithm to find MDEF-based outliers requires computation of $\frac{1}{2\alpha r}$ range queries, one for each of the intervals as shown in Figure 3. The complexity for one range query is $O(d|R|)$, and hence the overall complexity is $O(\frac{d|R|}{2\alpha r})$. For 1-dimensional data, the running time is $O(\frac{log|R|+|R'|}{2\alpha r})$, where R' is the set of kernels that intersect the query. □

8.1 Updating the *Global* Estimator Model

The naive approach to maintaining an updated *global* estimator model at every sensor at the leaf level, is to transmit

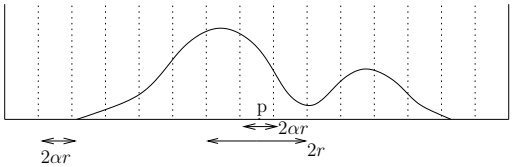


Figure 3: Estimating neighborhood count with a probability density function. For a new observation p, the number of $\alpha r - neighbors$ is estimated with range query $N(p, \alpha r)$. The domain of the observations is divided into intervals of width $2\alpha r$, and the number of points in the i^{th} interval is estimated with a range query $N(\alpha r(2i-1), \alpha r)$.

all the measurements to the leader at the highest level (where the leader computes the *global* estimator model, and transmits it back to all the leaf sensors).

However, this approach is very expensive in terms of communication overhead. Therefore, we propose the following scheme, where the number of updates are significantly reduced. For every new observation that is added to the local estimator model at the lowest level sensors, the sensors transmit the observation to their leaders with probability f. The leaders receive the observations and transmit them in-turn to their parents with probability f. When a new observation is added to the kernel sample maintained at the leader node of the highest level, this update is communicated to the lowest level sensor nodes via the intermediate leaders. Thus, for every new observation a sensor at the lowest level sends, it receives $(fl)^n$ updates to its global estimator, where n is the number of levels in the hierarchy and l is the average number of children per a parent node. The algorithm described in Figure 4 follows this scheme.

Let us assume a *centralized* method, where all the observations from all the sensors are communicated to the leader at the highest level, where the MDEF-based outliers are detected. We observe that the above scheme of sending the measurements with probability f can perform better than the centralized approach, only if f is small.

The communication overhead can be further optimized if we allow the leaders at each of the levels to update the children only when their estimator model has *significantly* changed. With this scheme, a parent sensor computes the distance between the estimator model that was last sent to the children, and its current estimator model. If the distance is greater than a pre-specified value, it sends the current estimator model to the children. In this way, the sensors at the leaf level receive fewer updates, particularly when the distribution of the underlying measurements is stationary.

Since this technique for outlier detection is based on local conditions, it is more accurate than the distance based outlier detection method. However, maintaining a copy of the *global* distribution function at all the sensor nodes requires frequent updates across the sensors and their parents. For a system with update probability f, there will be $O((fl)^n)$ messages communicated for every new observation, per sensor. We compare the number of messages communicated for this technique, with that of $D3$, in Section 10.3.

9. OTHER APPLICATIONS

An accurate online approximation of the probability density function allows us to solve a number of problems in a sensor network.

Online Query Processing: One category of problems is to

Algorithm D3

```
1 let W^w and W^b be the sliding windows
      of the leaf and parent nodes;
2 let R^w and R^b be the samples on W^w and W^b;
3 let σ^w and σ^b be the standard deviations on W^w and W^b;
4 let f be the fraction of the sample propagated from a child
      to its parent;
5 procedure D3()
6     assign one leaf node to each one of the input streams;
7     configure all parent nodes in a hierarchy on top of
      the leaf nodes;
8     initiate ParentProcess() for each parent node;
9     initiate LeafProcess() for each leaf node;
10    return;
11 procedure LeafProcess()
12     when a new value S(i) arrives
13         update R^w, σ^w;
14         if (S(i) included in R^w)
15             send S(i) to parent with probability f;
16         IsOutlier(R^w,σ^w,S(i));
17         if (S(i) is an outlier)
18             report S(i) as an outlier;
19             send S(i) to parent;
20     return;
21 procedure ParentProcess()
22     when a new message from a child node arrives
23         if (message is new outlier P)
24             IsOutlier(R^b,σ^b,P);
25             if (P is an outlier)
26                 report P as an outlier;
27                 send P to parent;
28         if (message is new value from child l)
29             update σ^b and R^b
30             if (the new value is included in R^b)
                   send new value to parent with probability f;
31     return;
32 procedure IsOutlier(sample R, stddev σ, point P)
33     use R and σ to estimate N(P, r);
34         if (N(P, r) < t)
35             mark P as an outlier;
36     return;
```

Algorithm MGDD

```
1 Let R^g and σ^g be the sample and standard
deviation at the leader of the highest level;
2 Rest of the notations are same as in algorithm D3
3 procedure MGDD()
4     assign one leaf node to each one of the input streams;
5     configure all parent nodes in a hierarchy on top of
      the white nodes;
6     initiate ParentProcess() for each black node;
7     initiate LeafProcess() for each white node;
8     return;
9 procedure LeafProcess()
10    when a new value S(i) arrives from stream
11        update R^w, σ^w;
12        IsOutlier(R^g,σ^g,S(i));
13        if (S(i) is added to R^w)
14            send S(i) to parent with probability f;
15    when a new update of R^g and σ^g is received
16            update R^g and σ^g
17    return;
18 procedure BlackProcess()
19     when a new message from a child node arrives
20         if (message is added to R^b)
21             send message to parent with probability f;
22             if (leader at the highest level)
23                 send updates of R^b and σ^b to all the children;
24     return;
25 procedure IsOutlier(sample R, stddev σ, point P)
26     use R and σ to estimate N(p, αr) and
       N(αr(2j − 1), αr) for each interval j;
27     if (isMDEFOutlier())
28             mark P as an outlier;
29     return;
```

Figure 4: Outline of the algorithms.

Dataset	Min	Max	Mean	Median	StdDev	Skew
Engine	0.020	0.427	0.410	0.419	0.053	-6.844
Pressure	0.422	0.848	0.677	0.681	0.063	-0.399
Dew-point	0.113	0.282	0.213	0.212	0.027	-0.182

Figure 5: Statistical characteristics for the real datasets.

provide approximate answers to range queries with both spatial and temporal constraints. These are queries of the following form. "What is the average temperature in region (X, Y) during the time interval $[t_1, t_2]$?". In such cases, the sensors can estimate the density model for the observations during the specified time interval and answer the queries based on the estimated model.

Finding Faulty Sensors: Another important application is online detection of faulty sensors. Examples include queries of the form: "Give a warning when the values of a given sensor are significantly different from the values of its neighbors over the most recent time window W", or queries of the form: "Give a warning if the number of outliers in a given region exceeds a given threshold T over the most recent time window W". Such queries can be very useful for monitoring the network for signs of malfunctioning sensors or for signs of possible intrusion. With our approach, a parent sensor can compute the difference between the estimator models received from its children, to determine if any of them is faulty.

In the interest of space, we defer further discussion of these applications to the full version of this paper.

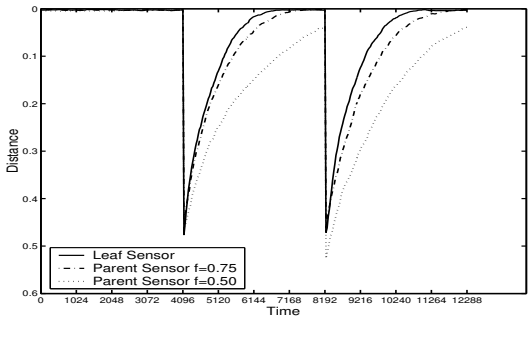
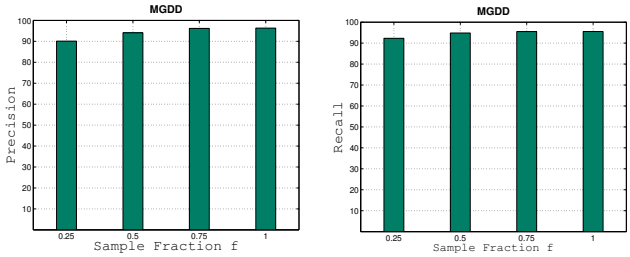


Figure 6: Difference between real and the estimated data distributions, at leaf and parent level.



(a) (b)

Figure 8: Performance of MGDD with varying sample fraction f (1-d synthetic data, Kernel approach).

10. EXPERIMENTAL EVALUATION

Implementation. We built a simulator to evaluate our framework, implemented on top of the TAG [32] simulator. Specifically, we use the TAG simulator infrastructure in order to define the topology of the network and the type of messages exchanged, to disseminate queries, and to gather statistics. We also made the necessary modifications to enable the hierarchical organization of the nodes in the sensor network. TAG, by default, builds a different spanning tree each time a new query is injected into the network. Instead, our simulator initiates a continuous query on every node. This enables us to program the outlier detection algorithm in the whole network. Then, we define the network hierarchy as described in Section 2. The nodes run the algorithms on top of the hierarchy imposed on them. More specifically, we implemented the following components: (i) *chain-sample*, which maintains a running sample of the sensor readings in the window, (ii) *variance estimator*, which maintains a running estimate of the standard deviation of those values, (iii) *kernel density estimator*, which is used to approximate the data distributions, and also contains the required machinery for using these approximations to answer queries, (iv) *distributed deviation detection* algorithm, to demonstrate the applicability of our first approach. (v) *MDEF-based deviation detection* algorithm, to demonstrate the applicability of our second approach.

Our implementation required $5,000$ lines of Java code. However, the code that implements our algorithm has very small footprint. For instance, the kernel density estimation and outlier detection modules (that is, the code that would have to run on a sensor to implement our algorithm) required a total of 150 lines of Java code.

Datasets. In our experiments we used a variety of synthetic and real datasets. The synthetic datasets are time sequences that are $35,000$ observations long each, and their values were normalized to fit in the $[0, 1]$ interval. Each dataset is a mixture of three Gaussian distributions with uniform noise; the mean is selected at random from $(0.3, 0.35, 0.45)$, and the standard deviation is selected as 0.03, so that it doesn't cover the entire space. Subsequently, we add 0.5% (of the dataset size) noise values, uniformly at random in the interval $[0.5, 1]$.

The first set of real datasets records the operation of an engine reported every 5 minutes by 15 sensors. The measurements span from June 1st 2002 to December 1st 2002, and form time sequences of $50,000$ values.

The second set of real datasets represents measurements of various natural phenomena, reported by a number of sensors in the Pacific Northwest region [2]. The datasets include measurements of atmospheric pressure, dew-point, temperature, solar radiation, and others. They span a two year period, and form time sequences of $35,000$ values. We report results where the observations at the sensors are streams of pairs (*pressure*, *dew-point*).

Note that in all the experiments we report, each sensor sees a *different* set of data. The characteristics of the real datasets are given in Figure 5.

Measures of Interest. We evaluate the accuracy of our methods in detecting distance-based outliers [28, 38] and MDEF-based outliers [36]. We use two measures, namely *precision* and *recall*, defined as follows. *Precision* represents the fraction of the values reported by our algorithm as outliers that are true outliers. *Recall* represents the fraction of the true outliers that our algorithm identified correctly.

Comparisons. We use offline algorithms to compute the true outliers for each instance of the sliding window. In order to identify the true distance-based outliers, we use the *BruteForce-D* algorithm. This algorithm accesses all $|W|$ points in the sliding window, and for each one of them, computes its distance to all the other points, guaranteeing to

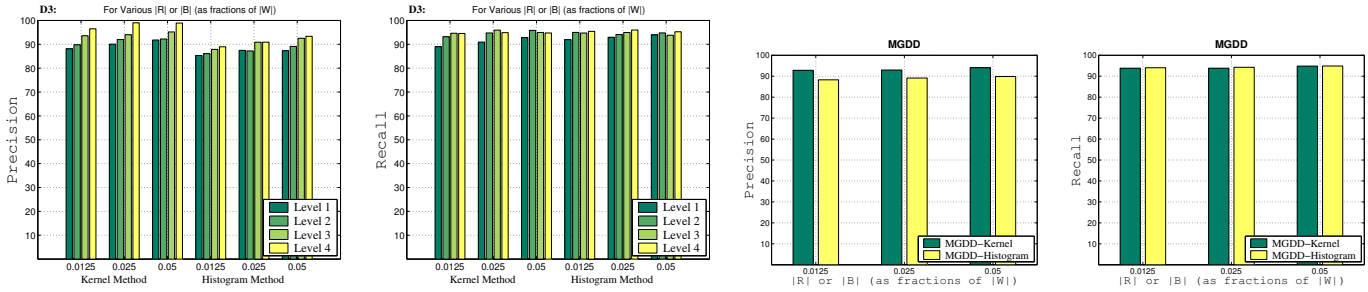


Figure 7: Precision and recall for the 1-d synthetic data with Kernel and Histogram approaches, when varying the memory of the representation ($|R|$ or $|B|$).

find all the true outliers. The naive implementation[2] of the *BruteForce-D* algorithm has time complexity $O(d|W|^2)$, where d is the dimensionality, and $|W|$ is the size of the sliding window. To identify the true MDEF-based outliers, we use the *aLOCI* (we will refer it as *BruteForce-M*) algorithm [36], which approximates the average neighborhood count and the standard deviation of neighborhood count based on an *interval count* over the measurements in the sliding window. We also compare the performance of our approach with *histograms*, a common method for approximating distributions. For the leaf level sensors, we compute equi-depth histograms of $|B|$ buckets by accessing all $|W|$ values in the sliding window. At a higher level sensor s, we compute a histogram of $|B|$ buckets by accessing *all the values* in the sliding window of all the leaf level sensors rooted at s. This approach favors the histogram technique over the kernel method we propose, in which higher level sensors only see a *fraction of the sample* of their children (see Section 5.1). We set $|B| = |R|$, in order to ensure comparable memory usage by our approach and the histogram approach. We note that this brute-force approach of computing histograms is prohibitive in terms of both the communication and computation overhead. In addition, the computation is *not* done in an online fashion. However, this method gives an upper-bound for any dynamic version and thus serves as a good measure for comparing the results of the kernel based approach.

10.1 Accuracy of Data Distribution Estimation

In the first set of experiments, we evaluate the accuracy of the kernel density estimators in our setting. More specifically, we measure the distance between the true probability density function of the streaming sensor data and our estimate of this function. The window size is $W = 10240$ and the sample size is $|R| = 1024$. We consider Gaussian distributions and vary the underlying distribution after every 4096 measurements (from $\mu = 0.3$, $\sigma = 0.05$ to $\mu = 0.5$, $\sigma = 0.05$) to measure the latency with which the sensors adjust to the changes in distribution. We also measure the distance between the true probability density function and the estimated function at a parent sensor, for various values of f. In all the experiments, the distance is computed based on the JS-divergence (Section 6), and the distance ranges from 0 to 1.

Figure 6 shows that our approximation is very close to the real data distribution, with a maximum distance of 0.0037 when the distribution of the measurements remains stable. The maximum distances at the parent sensor for $f = 0.5$ and $f = 0.75$ are 0.0051 and 0.004. When the distribution changes, the estimate of the streaming data deviates considerably from the true distribution. This is expected, because most of the measurements in the current sliding window are generated from the earlier distribution function. However, with time, the distance reduces, and is within 0.1 with latency of 2500 measurements. We observe that for the parent sensor, the latency decreases with increasing f.

[2]There exist more efficient implementations [28] that have to operate offline.

10.2 Accuracy of Outlier Detection Mechanism

In the following paragraphs we evaluate the accuracy with which our algorithms detect outliers, under different parameter settings. The setup of the experiments involves 48 nodes, organized in a hierarchy with 3 levels. There are 32 nodes in the lowest level, each producing a different stream of data, and two levels of leaders above them. All the results we report are averages over 12 runs of each experiment, and in all cases the number of outliers was between $40 - 80$. In our experiments, we varied the size of the sample $|R|$ used by the kernel estimators, the fraction of the sample f that each node propagates to its leader, and the size of the sliding window $|W|$. Unless otherwise noted, the default values for the parameters are $|W| = 10,000$, $|R| = 0.05|W|$, and $f = 0.5$. For the distance-based outliers, we are looking for $(45, 0.01)$-outliers i.e., an observation is an outlier if the number of neighbors within a radius of 0.01 is less than 45. For the MDEF-based outliers, we set the sampling neighborhood $r = 0.08$ and the counting neighborhood $\alpha r = 0.01$. In all our experiments for finding MDEF-based outliers, we consider $k_\sigma = 3$.

In every case, we measured the precision and the recall, and the results for the synthetic datasets are depicted in Figures 7 and 8 (1-d data), and 9 (2-d data). Overall, $D3$ algorithm with kernel approach achieves around 94% precision and 92% recall. With $MGDD$, where we only detect outliers in the first level of the hierarchy, the precision and recall average around 94% and 93%, respectively. For both methods, both metrics are over 90% with the right choice of parameters, indicating that they are very promising approaches. We note that the set of outliers detected with $D3$ and $MGDD$ are *different*, due to their definitions, and therefore, we should not compare the accuracy of the methods against one another.

In Figure 7, we also depict the performance of $D3$ and $MGDD$ when using histograms instead of kernels. The results demonstrate that in all cases the kernels approach is as good as histograms, and in many cases (for the precision metric) kernels outperform histograms. This is true despite the fact that, as we discussed earlier, in our implementation we have favored the histogram approach. We expect that any

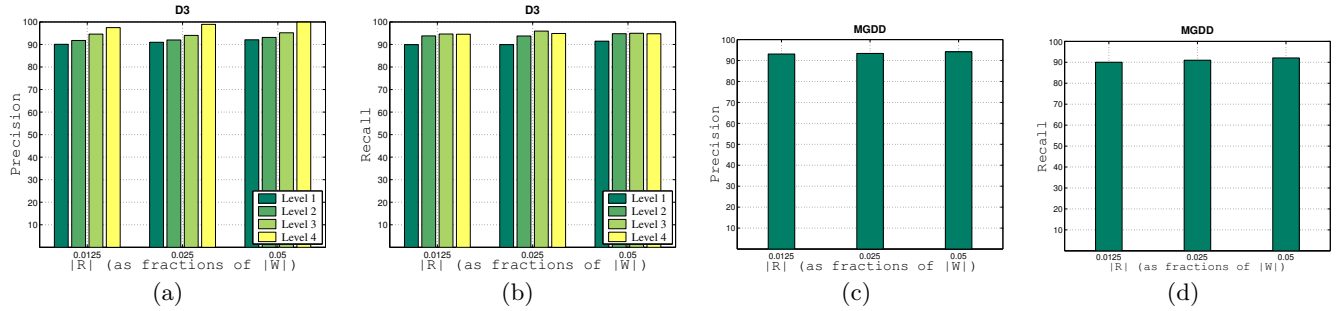


Figure 9: Precision (a, c) and recall (b, d) when varying $|R|$ (2-d synthetic data, Kernel approach).

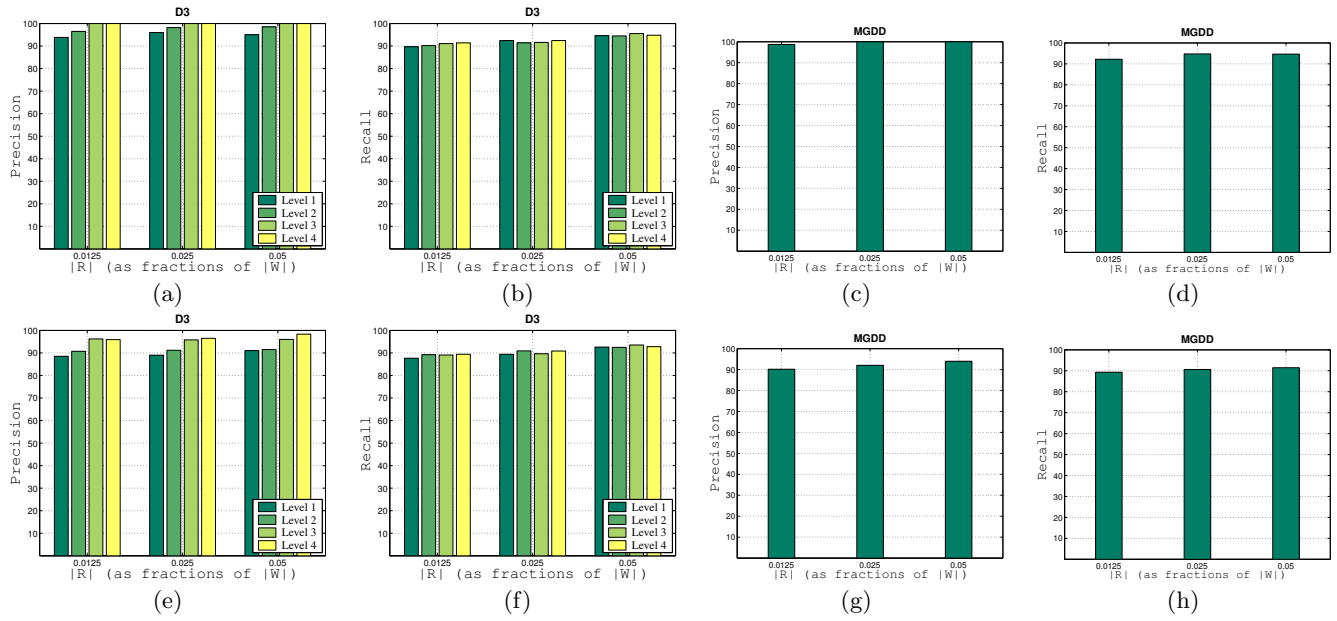


Figure 10: Precision and recall when varying the sample size $|R|$, for the real datasets (Kernel approach): 1-d *engine dataset* (upper graphs) and 2-d *environmental phenomena* dataset (lower graphs).

similar online techniques will perform at most as good. In the next paragraphs we focus our discussion on the kernels approach.

The experimental results indicate that the precision for $D3$ gets better as we go up in the network hierarchy, for both the 1-d (Figures 7(a)) and the 2-d data (Figure 9(a)). This does not come as a surprise, because of the way the algorithm works. Recall that the nodes in the lowest level of the hierarchy have to examine every single data measurement in order to determine if it is an outlier or not. Then, they only transmit to their leaders the values they have identified as outliers. This means that the nodes in each subsequent, higher, level has to examine fewer values. Moreover, these values have a very high probability of being outliers (see Theorem 3). Hence, the algorithm reports fewer false positives (i.e., precision increases) as we move up in the hierarchy.

The same observation is, in general, true for the recall, as well (Figures 7(b) and 9(b)). This is due to the fact that the number of outliers drops for the higher levels (a consequence of Theorem 3). We also note that recall diminishes in some cases for nodes in upper levels. This happens because these nodes only examine the values that their children have already identified as outliers. However, the children nodes may have missed some of the true outliers, which will consequently cause the leader node to miss them.

When we increase the sample size, the performance of the $D3$ algorithm improves slightly (Figures 7(a, b) and 9(a,b)). The $MGDD$ technique is less affected by the change of the sample size (Figures 7(c,d) and 9(c,d)), but its performance improves as the sample fraction f increases (Figure 8). This is expected, because f determines the rate at which the observations are sent from the children nodes to their parent, and thus influences the frequency with which the *global* estimators at the leaf sensors are updated. The same behavior when varying the sample fraction f is observed with $D3$ as well, but we omit the relevant graphs for brevity.

We have also conducted experiments where we varied the size of the sliding window $|W|$ between $10,000$ and $20,000$ values. The results show that the performance of our techniques remains relatively stable as we vary this parameter. In the interest of space, we report these results in the full version of the paper.

Figure 10 depicts the results we obtained with the real datasets, engine (1-d data) and environmental phenomena measurements (2-d data), with our kernel based approach. With $D3$, we were looking for $(100, 0.005)$-outliers. For the $MGDD$ technique, we set $r = 0.05$ and $\alpha r = 0.003$. The graphs show that the trends for these datasets remain the same as with the synthetic ones. Both algorithms averaged around 99% precision, and 93% recall for the *engine* measure-

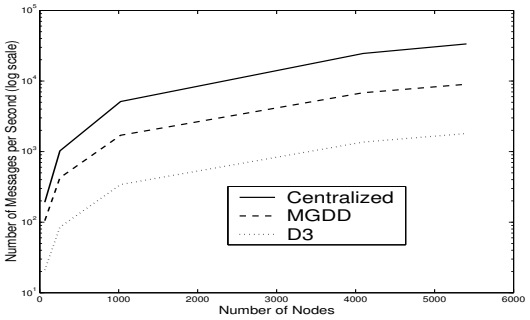


Figure 11: Number of messages in network per second (log-scale), while varying the number of sensors.

ments, which is better than the performance of the algorithms on the synthetic datasets. This is due to the *smooth* nature of the data set, except for the measurements observed from October 28th to November 1st, where a major failure was detected in the systems and they reported deviating values. The results for the environmental 2-d dataset were comparable to those obtained with the synthetic 2-d dataset.

10.3 Memory and Communication Costs

In order to verify the efficiency of our technique, we ran experiments to measure the maximum amount of memory required by the $D3$ algorithm per node. There are two components of our algorithm that affect the memory consumption: sample maintenance and variance estimation. The memory requirement of the former is upper-bounded by $O(d|R|)$, and of the latter by $O(\frac{d}{\epsilon^2}\log|W|)$.

We ran experiments using the real datasets, and assuming a 16-bit architecture, i.e., 2 bytes per number. We varied the size of the sliding window $|W|$ (10000-20000), as well as the sample fraction f (0.25-5). The experiments showed that in all cases the actual values of the maximum memory consumption of the variance estimation procedure is around 55%-65% less than the theoretic upper bound.

We also ran experiments in order to quantify the number of messages that are generated, by scaling up the number of the nodes in our testbed. We compare our algorithms, *D3* and *MGDD* against the centralized approach. For our approach we take into account the number of messages generated due to the incremental sample propagation. We do not account for the messages sent when a local outlier is identified, since these are infrequent. We assume that each sensor generates one reading every 1 second. The size of the window $|W|$ was set to 10240, the sample size $|R|$ was set to 1024, and the sample fraction f was equal to 0.25. Figure 11 shows the number of messages generated per second (in log scale), while scaling up the number of nodes. As expected, the $D3$ approach gives better savings compared to both $MGDD$ and centralized. We observe that the $D3$ algorithm requires approximately two orders of magnitude fewer messages, and hence the best method with respect to optimizing communication cost.

11. RELATED WORK

Madden and Franklin [31] present a framework for the efficient execution of queries in a sensor network. The problem of evaluating aggregate operators in a sensor network is addressed by Madden et al. [32]. Yao and Gehrke [45] investigate the problem of query processing in sensor networks. In a complementary study, Bonfils and Bonnet [9], describe an algorithm for mapping a tree of query operators on the sensor network.

A recent study [16] proposes a sensor data acquisition technique, based on models that approximate the data with probabilistic confidences. This general technique results in reduced communication costs, without sacrificing much of the accuracy [15]. However, any special characteristics of the data distribution, such as periodic drifts, have to be explicitly encoded in the space of models considered. In our work, we describe a more general technique, which can efficiently overcome this limitation. Moreover, we observe all the data values, and can therefore reason about outliers, whereas the above technique aims at minimizing the cost of making *some* observations that will ensure the user-defined probabilistic confidence thresholds are met.

A framework for modeling sensor network data is also proposed by Guestrin et al. [20]. The goal in this approach is for the nodes in the network to collaborate in order to fit a global function to each of their local measurements. This is a parametric approximation technique, and as such, requires the user to make an assumption about the number of estimators required to fit the data. This model has more parameters to fit than the approach that we propose, where we only have to estimate a single parameter, thus reducing the requirements of in-network computation. Cormode and Garofalakis [13] describe a technique for approximate query tracking based on *sketches*. Their technique can efficiently operate in a distributed, online setting. Even though it can be generalized, it is mainly geared toward discrete domains and the unrestricted window model. In order to work for sliding windows, it would require to store all the values of the window, which is something we avoid doing in the framework we propose.

Greenwald and Khanna [19] study the problem of computing order statistics in a sensor network. Another recent study [41] addresses the problem of approximating the data distribution for computing order statistics, as well as range queries. There has also been work on predicting and caching the values generated by the sensors [35, 26], which can result in significant communication savings. Nevertheless, it is not obvious how to use this approach in our setting, since distance-based outliers require the computation of the number of neighboring values. It may be the case that values within the change detection threshold defined by the above approaches are outliers, and values outside this threshold are not. In addition, our model is designed to efficiently compute the distribution of a region, and therefore, identify outliers by combining the data from multiple sensors.

A similar approach for outlier detection in streaming data is described by Yamanishi et al. [44]. In contrast to our work, their method does not operate on sliding windows, but rather on the entire history of the data values, using an exponential forgetting factor for discounting the effect of the older values. Furthermore, the above approach is not geared towards a distributed environment, such as a sensor network.

There is extensive literature in the statistics community regarding outlier detection [6], as well as in the database community [3, 28, 38, 10]. However, none of these approaches is directly applicable to a sensor environment, either because they assume knowledge of the input data distribution, or because they are not tailored to operate online. There has been work on the special case of identifying outliers in streaming *time-series* data [37, 34]. Nevertheless, the significance of the temporal ordering is a major difference from the semantics of the problem we are considering in this study.

Recent work [22] gives an online technique to compute the JS divergence. This approach can be applied for some of the applications we consider, such as identifying faulty sensors but does not impact our algorithms for finding outliers.

12. CONCLUSIONS

In this paper, we study the problem of outlier detection in sensor networks. Outlier detection is very important in this context, since it enables the analyst to focus on the interesting events in the network. We propose a framework based on the approximation of the distribution of the sensor measurements. The techniques we describe operate efficiently in an online fashion. Moreover, they distribute the computation effort among the nodes in the network, thus better exploiting the available resources and cutting back on the communication and processing costs. We evaluated our approaches with a set of experiments with real and synthetic datasets. The experimental evaluation shows that our algorithm can achieve very high precision and recall rates for identifying outliers, and demonstrate the effectiveness of the proposed approach. As future work, we plan to evaluate our techniques in a real sensor network.

Acknowledgments: We would like to thank Samuel Madden for providing us the source code of the TAG simulator. The research of Vana Kalogeraki and Dimitrios Gunopulos is supported by NSF Grant 0330481.

References

[1] Crossbow Technology Inc. http://www.xbow.com/.
[2] Earth Climate and Weather, University of Washington. http://www-k12.atmos.washington.edu/k12/grayskies/.
[3] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. A Linear Method for Deviation Detection in Large Databases. In *KDD*, 1996.
[4] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling From a Moving Window Over Streaming Data. In *SODA*, 2002.
[5] Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining Variance And k-medians Over Data Stream Windows. In *PODS*, pages 234–243, USA, 2003.
[6] V. Barnet and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, Inc., 1994.
[7] Shai Ben-David, Johannes Gehrke, and Daniel Kifer. Identifying Distribution Change in Data Streams. In *VLDB*, Toronto, ON, Canada, 2004.
[8] Bjorn Blohsfeld, Dieter Korus, and Bernhard Seeger. A Comparison of Selectivity Estimators for Range Queries on Metric Attributes. In *SIGMOD*, 1999.
[9] B. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *IPSN*, 2003.
[10] M.M. Breunig, H.-P. Kriegel, R.T. Ng, and Jörg Sander. LOF: Identifying Density-Based Local Outliers. In *SIGMOD*, 2000.
[11] Paul G. Brown and Peter J. Haas. Techniques for warehousing of sample data. In *ICDE*, 2006.
[12] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate Query Processing Using Wavelets. In *VLDB*, 2000.
[13] Graham Cormode and Minos N. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24, 2005.
[14] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & sons, 1991.
[15] Amol Deshpande, Carlos Guestrin, and Samuel R. Madden. Using Probabilistic Models for Data Management in Acquisitional Environments. In *Proc. CIDR*, 2005.
[16] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB*, Toronto, ON, Canada, 2004.
[17] D. Ganesan, B. Greenstein, D. Estrin, J. Heidemann, and R. Govindan. Multiresolution storage and search in sensor networks. *ACM TOS*, 1(3):27–315, 2005.
[18] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *VLDB*, Rome, Italy, 2001.
[19] M.B. Greenwald and S. Khanna. Power-Conserving Computation of Order-Statistics over Sensor Networks. In *PODS*, 2004.
[20] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed Regression: an Efficient Framework for Modeling Sensor Network Data. In *IPSN*, Berkeley, CA, 2004.
[21] Sudipto Guha and Nick Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. In *ICDE*, pages 567–576, San Jose, CA, USA, 2002.
[22] Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *In Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, 2006.
[23] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. In *SIGMOD*, 2000.
[24] Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *ICDCS*, 2002.
[25] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal Histograms with Quality Guarantees. In *VLDB*, New York, NY, USA, 1998.
[26] A. Jain, E.Y. Chang, and Y.-F. Wang. Adaptive Stream Resource Management Using Kalman Filters. In *SIGMOD*, 2004.
[27] Ralph M. Kling. Intel Mote: An Enhanced Sensor Network Node. In *Workshop on Advanced Sensors, Structural Health Monitoring, and Smart Structures*, Kanagawa, Japan, 2003.
[28] E.M. Knorr and R.T. Ng. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *VLDB*, NY, NY, 1998.
[29] Lillian Lee. On the effectiveness of the skew divergence for statistical language analysis. In *Artificial Intelligence and Statistics 2001*, pages 65–72, 2001.
[30] J. Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Infor. Theory*, 37:145–151, 1991.
[31] Samuel Madden and Michael J. Franklin. Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data. In *ICDE*, 2002.
[32] Samuel Madden, Michael J. Franklin, and Joseph M. Hellerstein. TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *OSDI*, 2002.
[33] N. Malpani, J. Welch, and N. Vaidya. Leader Election Algorithms for Mobile Ad Hoc Networks. In *DIAL M Workshop*, 2000.
[34] S. Muthukrishnan, Rahul Shah, and Jeffrey Scott Vitter. Mining Deviants in Time Series Data Streams. In *SSDBM*, 2004.
[35] C. Olston, J. Jiang, and J. Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *SIGMOD*, 2003.
[36] S. Papadimitriou, H. Kitagawa, P. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral, 2003.
[37] Vasundhara Puttagunta and Konstantinos Kalpakis. Adaptive Methods for Activity Monitoring of Streaming Data. In *ICMLA*, 2002.
[38] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient Algorithms for Mining Ouliers from Large Data Sets. In *SIGMOD*, 2000.
[39] Dongmei Ren, Baoying Wang, and William Perrizo. Rdf: A density-based outlier detection method using vertical data representation. In *ICDM*, pages 503–506, 2004.
[40] D. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization*. Wiley & Sons, 1992.
[41] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and Beyond: New Aggregation Techniques for Sensor Networks. In *ACM SenSys*, Baltimore, MD, USA, 2004.
[42] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *SIGMOD Conference*, 2002.
[43] B. Warneke, M. Last, B. Liebowitz, and K. Pister. Smart dust: Communicating with a cubic-millimeter computer. *IEEE Computer Magazine*, pages 44–51, January 2001.
[44] Kenji Yamanishi, Jun ichi Takeuchi, Graham J. Williams, and Peter Milne. On-Line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, 2004.
[45] Yong Yao and Johannes Gehrke. Query Processing for Sensor Networks. In *CIDR*, Asilomar, CA, USA, 2003.
[46] Fan Ye, Haiyun Luo, Jerry Cheng, Songwu Lu, and Lixia Zhang. A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks. In *MOBICOM*, Atlanta, GA, USA, 2002.
[47] S. Zhao, K. Tepe, I. Seskar, and D. Raychaudhuri. Routing protocols for self-organizing hierarchical ad hoc wireless networks. In *IEEE Sarnoff Symposium*, 2003.