Efficient Online State Tracking Using Sensor Networks

M.Halkidi[1], V.Kalogeraki[2], D.Gunopulos[2], D.Papadopoulos[2], D.Zeinalipour-Yazti[3], and M.Vlachos[4]
[1]Athens U of Economics and Business, [2]UC Riverside, [3]U of Cyprus, [4]IBM T.J. Watson R.C.
Email: {mhalkidi, vana, dg, dimirtis, csyiazti}@cs.ucr.edu, vlachos@us.ibm.com

Abstract

Sensor networks are being deployed for tracking events of interest in many environmental or monitoring applications. Because of their distributed nature of operation, a challenging issue is to identify the aggregate state of the phenomenon that is being observed. This work presents an online mechanism for efficiently determining the overall network status, employing distributed operations that minimize the communication costs. Experiments on real data, suggest that the proposed metholology can be a viable solution for real world systems.

1 Introduction

Sensor networks consist of a large number of small, low-cost, wireless devices (referred to as sensor nodes) that are deployed over an area of interest. The nodes are equipped with sensing, communicating and data processing capabilities, which allow them to collect, exchange and process information about the monitored events in their environment [1, 2]. Sensor networks have been used successfully for information collection, environmental monitoring and tracking events of interest in fields, such as surveillance, agriculture, seismic activity detection, and healthcare.

We observe however that in many applications the user is mostly interested in a macroscopic description of the condition that the process or the phenomenon under observation is in, rather than the readings of the individual sensors. Consider for example the case where a network of temperature sensors has been deployed to monitor a building for fire hazard. Each room may contain more than one sensors, for fault-tolerance. Assume that the sensors are connected in a network forming a tree. A simple implementation would have each sensor periodically send its reading to the root of the tree (sink). When a sensor (or a set of sensors) sends back high readings, an alarm is sound. Note that, in order to make sure that a given sensor is not out of order, it has to communicate periodically. Through this simple application we begin to observe that not all transmitted information is important. For the majority of time, the actual temperature reading of any given sensor is not significant, as long as it falls within a reasonable data interval. In fact, for each sensor we can only have two important states, either there is fire in the vicinity of the sensor, or not. This is true to a large extend for the whole network. The status of the network can be described as “Safe” if there is no fire, or “Alarm” if there is. Similarly, we can add more states to describe more accurately the status of the network if we need more discrimination, such as “Alarm in Floor 3”, etc.

In this paper we consider the problem of efficiently computing an accurate aggregate picture of the *condition* of a sensor network. We define the condition of the sensor network to be the *union* of the conditions (that is, the readings) of the individual sensors. Equivalently, we can represent the condition of the network as a point in a K-dimensional space where K is the number of sensors and each dimension represents the reading of a sensor node. We use the notion of network *states* to abstract the condition of the network. We use a clustering process to put together similar sets of readings; we consider each such cluster of readings as a distinct “state” of the network. The clustering results are then used to define a partitioning of the multidimensional space of sensor readings. Each sensor is then provided with the projection of this partitioning to its dimension. This *learning* process is online and runs continuously to properly define (in the beginning) and update (later on) the states of the sensors as more sensor readings arrive.

Then, the system runs an online *monitoring* process that discovers the status of the observed phenomena by monitoring the conditions of the individual sensor nodes. The sensor nodes are organized in a hierarchical structure in which the sensors inform each other about possible state changes. These updates are propagated to the root using an efficient aggregation procedure.

Once a new state has been detected in the whole (or parts of the) network, the user can use additional querying mechanisms (e.g. TAG [3]) to retrieve more details from specific sensor nodes. For example our mechanism can alert the user that there is fire at North to Northwest. Then, the user can query the local sensors to exactly pinpoint the fire.

Our approach has multiple benefits:

(1) We improve the *scalability* of the system over central-

ized approaches by making use of in-network processing to identify the network state.
(2) We provide an *abstraction* of the individual readings of the sensors to a set of general states. This reduces the communication and computation overhead: each sensor transmits an update message less frequently, because small variations in its local values are not likely to change the overall status of the network.
(3) We provide *robustness* by developing a mechanism that can identify the correct network state even in the presence of network faults.

2 Related Work

There is a large body of research in distributed detection and estimation, including the topic of multi-sensor fusion. In [4] the binary detection problem with multi-message communication is considered. Alhakem and Varshney [5] study a distributed detection system with feedback and memory. That is, each sensor not only uses its present input and the previous decision from the fusion center, but it also uses its own previous inputs. Swaszek and Willet propose a more extensive feedback approach [6]. The basic idea is that each sensor makes an initial binary decision which is then distributed to all the other sensors. The goal is to achieve a consensus on the given hypothesis through multiple iterations. There is currently significant work in the sensor networks' field that aims to address the problem of efficient management of the data in the network. In the context of the COUGAR project [7] an architecture of a distributed data management system in sensor networks is proposed. Issues related to the distributed in-network processing, query optimization and query languages are also addressed. Another related system is TinyDB [8] which implements the TAG framework [3]. It is a query processing system for extracting information from a network of motes.

In [9] a distributed monitoring mechanism for sensor networks is proposed. This approach aims to detect if the sensors are alive as well as potential failures in the network. Also [10] presents techniques for reliable transmission of data from nodes to base station. Compression techniques for transmitting data are also proposed.

Our work improves such general techniques by using a learning phase designed to optimize the system performance while minimizing the load on the sensor network resources. It aggregates the individual sensors' knowledge about network status.We also consider the effects of lost or corrupted messages on the performance of the detection/estimation algorithm.

3 Sensor Network Architecture

We assume a sensor network that has been deployed to observe a process or a phenomenon, and in which the nodes communicate with a wireless network. There is a base station (further referred to as *sink*) which monitors the status of the process, and periodically issues a query to discover the current state. The sink broadcasts a query in the network and organizes the sensor nodes into a tree or a Directed Acyclic Graph (DAG, similar to [11]) in which the root is the sink. This operation can be done for each query, using the following simple mechanism: The query message has a counter that is incremented with each retransmission and counts the hop distance from the root. Thus, each node is assigned to a level equal to the node's hop distance from the root. Each node selects some of its neighbors that have a smaller hop distance from the sink (root) to be its parents in the tree. The number of parents selected is a user specified parameter. When the choice is a large number, the expected accuracy of our mechanism for a given number of failures improves, at the cost of using more resources in the form of more messages transmitted.

Each leaf node in the tree produces a message with its state estimates and forwards this message to its parents. The non-leaf nodes receive the state messages of their children and combine them based on an aggregation function. Then, they submit the new partial results into their own parents. This process runs continuously until the total result arrives at the root where the final decision for the overall system state is made. The sink of the network does not need to be static but periodically a new node can be considered to be the sink.

4 Network Status Monitoring

Our technique consists of two processes, summarized as follows:

1. *Learning Process*. This process runs continuously as new sensor data arrives in form of streams. The goal is to define the states of the network and update them properly as new measurements arrive. It uses the following two *online* mechanisms:
   a) Clustering sensors' readings using user constraints to define the sensor network states.
   b) Extracting rules that describe the state of the network given the readings of the sensors.
2. *Status Monitoring Process*. This is an *on-line process* at which the sensor nodes are collaborating to update the status of the overall network by applying the rules to the readings of the individual sensors.

4.1 Defining the states of a sensor network

The definition of the states of the sensor network is done by the learning process of our approach. In some cases we can assume that the states are predefined or a domain expert has given the information about the states of interest based on the sensor readings. However, there may be patterns in the sensors' readings that are not previously known or the user ignores, and thus an automated process is necessary. A straightforward approach would be to use historical (previously collected) data from the sensor network, and to try to identify segments in the sensor reading space that show similar behavior. Such an approach, which reduces the state discovery problem to a simple clustering problem, has a number of problems: (1) the discovered clusters may have no physical meaning, or intuitive description, (2) the clusters may not correspond to the user's idea on what the network states should be, (3) since we have no control on the data used for the clustering, some situations may be under-represented, or not represented at all, forcing the clustering algorithm to miss the corresponding states altogether.

To tackle the above problems we adopt an approach that defines the potential states of the network not only using historical data of the sensor network, but also by applying the knowledge of the domain experts. Moreover since a large volume of sensor data is arriving continuously, it is either unnecessary or impractical to store the data in some form of memory; we focus instead in giving an online state discovery algorithm.

Satisfying User Constraints: The user inspects the results of the clustering process and if the discovered clusters do not match the states that the user wants, he or she attempts to improve on the clustering by adding constraints that have to be satisfied by the clustering. We adopt constraints of very simple form, making the constraint-setting process easier and more intuitive.

We assume we have a set of K-dimensional sensor network conditions $X = \{x_1, \dots, x_m\}$. The user specifies a set $\mathcal{S}$ of *must-link* constraints, and a set $\mathcal{D}$ of *cannot-link* constraints:

$\mathcal{S} : (x_i, x_j) \in X$ if x_i and x_j must belong to the same cluster
$\mathcal{D} : (x_i, x_j) \in X$ if x_i and x_j cannot belong to the same cluster

Then the goal is to learn a distance metric between the points in X such that when the data are clustered using an existing clustering (K-Means in our case), the resulting clustering satisfies the given constraints.

The problem of semi-supervised clustering comes up in many domains, and has inspired many recently proposed algorithms [12, 13]. In [14] we presented a framework for incorporating user constraints to the clustering result. We stress, however, that the choice of the clustering algorithm used is orthogonal to the rest of our technique. Indeed, the technique would work with *any* state discovery method, including having the user define the states *manually*.

Our technique evaluates a clustering, C_i, of a dataset in terms of its accuracy with respect to the given constraints and its validity based on well-defined cluster validity criteria. It is given by the equation:

$$QoC_{constr}(C_i) = w \cdot Accuracy_{S\&D}(C_i) + (1 + S\_Dbw(C_i))^{-1} \quad (1)$$

The first term of QoC_{constr} assesses how well the clustering results satisfy the given constraints. It is the proportion of the $\mathcal{S}\&\mathcal{D}$ constraints that are satisfied in the C_i clustering. The second term of QoC_{constr}, is based on a cluster validity index, S_Dbw [15]. A more detailed discussion on the quality measure QoC_{constr} is presented in [14].

The fundamental idea of our approach is to learn a weighted Euclidean distance metric so that the user constraints are satisfied. Then a clustering algorithm (K-Means in our case) using that metric is applied to the set of sensor readings to define the clusters. Specifically, considering a distance metric of the form,

$$d_A(x,y) = \sqrt{(x-y)^T A (x-y)},$$

we aim to define the A matrix so that both the *must-link* and *cannot-link* constraints are satisfied. Then the problem of learning a distance measure to respect a set of constraints (as defined above) is reduced to solving an optimization problem [12].

4.1.1 Online Clustering based on user constraints

As we discussed above the readings of sensors arrive in the form of a data stream and it is therefore impractical to store all the sensor data in memory. We propose to use a *novel* two step approach for clustering data streams while utilizing user constraints.

The first step of our approach can be considered as a preclustering process where we keep only a part of the points that have arrived up to a given point. The goal is to drop points that are similar to previous processed points, thus producing a good representation of the data seen so far. This goal is achieved by using a hierarchical stream clustering mechanism (inspired by the approach of [16]) that clusters all the points we have seen so far into m clusters. The size of m is a parameter in our technique, however there is a trade-off since increasing m increases the accuracy of the technique, but decreasing it improves the scalability and efficiency of the next step. The outline of Step I is as follows:

1. Let $\mathcal{X}$ be the set of the current representatives, and S, D sets of constraints on these representatives.
2. When a new set of points $\mathcal{X}'$ arrives, if $|\mathcal{X}| + |\mathcal{X}'| \leq m$, $\mathcal{X} = \mathcal{X} \cup \mathcal{X}'$, otherwise:
    (a) Use a clustering algorithm to cluster $\mathcal{X} \cup \mathcal{X}'$ (e.g. the effi cient hierarchical algorithm of [16]) into m clusters.
        i. Use the weighted Euclidean distance derived in Step II to compute distances.
        ii. In the operation of the hierarchical algorithm do not merge clusters that are in the same constraint in $\mathcal{D}$ (dissimilarity constraint).
        iii. When the hierarchical clustering algorithm merges clusters, update the constraints in $\mathcal{S}, \mathcal{D}$ the original clusters appear in.

(b) Find the m centroids of these clusters. Let X_{new} be the set of these centroids.

(c) **If** $|X_{new} - \mathcal{X}| \leq \theta$ **then** $\mathcal{X} \leftarrow X_{new}$,
Else Go to Step II.

Each time a new reading, p, arrives, we run Step I. If the result of Step I is a change in the set $\mathcal{X}$, we apply Step II, to find if and what changes have to be made to the current clustering of sensor readings (i.e. sensor states). There are two ways to incorporate a new point: (i) we can add the new point to an existing cluster and change that cluster's description, or (ii) we have to assign the new point to a new cluster. In this process, we possibly redraw some of the original clusters, taking also into account all user's input so far. We note here that in practice Step II can be run when a larger number of updates in $\mathcal{X}$ have been identified, thus running the update process in batches.

The result of Step II is a clustering of the points in $\mathcal{X}$. The state update process aims to tune the dimension weights so that the data are transform to a new space where a clustering algorithm (here, K-Means) can find the 'best' clustering in terms of i) the user preferences, and ii) objective cluster validity criteria. The outline of the update algorithm (Step II) is given below:

1. Given a new point p in $\mathcal{X}$, a current clustering C, and a user parameter ϵ.
2. **If** *there is no cluster $c_i \in C$ so that p falls in the boundaries of c_i* **then**

   (a) *Assess the quality of the current clustering, C*, calculating the value of the quality measure, $QoC_{constr}(C)$, we have adopted.

   (b) *Assign p to the cluster with the closest centroid* so that the user constraints are not violated.

   (c) *Calculate the value of quality measure (QoC_{constr})* for the new clustering let, C'.

   (d) **If** $(|QoC_{constr}(C) - QoC_{constr}(C')| > \epsilon)$ **then** *Check* the following cases

      i. *Merge the closest pair of clusters*, and add a new cluster containing p. Evaluate the quality of the new clustering, C_1.

      ii. *Create a new cluster* that contains only p. Evaluate the quality of the new clustering, C_2.

      iii. *Compare* $QoC_{constr}(C')$, $QoC_{constr}(C_1)$, $QoC_{constr}(C_2)$, and select the new clustering to be the one that corresponds to the best value of the quality measure, i.e. $C = C_k \in \{C', C_1, C_2\}$ such as

$$QoC_{constr}(C_k) = max\{QoC_{constr}(C'),$$
$$QoC_{constr}(C_1), QoC_{constr}(C_2)\}$$

   (e) *Present* the currently selected clustering, C, to the user

   (f) **While** *the user is not satisfied* **do**

      i. Let user give feedback and update the set of must- and cannot-link constraints ($\mathcal{S}\&\mathcal{D}$).

      ii. *Tune* the distance measure and *redefine* C using a semi-supervised clustering approach.

4.2 Converting clusters to states

Generally, the clustering technique contributes in compressing the information included in sensor readings. We use each cluster to define a region in the K-dimensional space of sensor readings that represents a state. To create state descriptions that can be applied to the readings of individual sensors we find the minimum bounding hyperrectangle (with axis aligned hyperplane) for each cluster. Now a range of values can represent a state, thus making the state description more compact and allowing for a representation with a simple numbering scheme. Moreover, instead of a sensor transmitting continuously its sensor readings, only its current state needs to be transmitted, reducing the communication bandwidth and ultimately leading to drastic energy savings.

To compute the description of a state, we simply compute the projection of the points in the cluster to the original K dimensions, and in each dimension we find the minimal interval that encompasses all the projections. The state is then the intersection of the K one-dimensional intervals.

One problem we have to consider is that after we project clusters to the original dimensions, we may get overlaps between the regions of different states (see Figure 1). To solve this i) we find all the ambiguous regions which can easily be done using a R-tree [17] structure. Since we already have rectangular representations of the clusters the R-tree gives us the rectangular intersections, ii) each intersection becomes a new state, iii) we ask the user to decide if these new states can be merged with the existing ones.

Extracting Rules for sensor network states. Based on the results of the clustering process we extract a set of rules which describe the conditions based on which the states of the network can be defined. The rules are sensor specific and describe what the specific sensor can deduce about the network state based on its local information (readings).

This is done by taking the projections of the clusters onto each original dimension, and finding the intersections of the projections. Given n clusters (states), this creates at most $2n + 1$ distinct, non-overlaping, intervals. We create one rule for each interval. Since the intervals are non-overlaping, the application of the rules is unambiguous: for each reading one rule applies. The condition (left hand side of the rule) refers to a range of sensor node readings while the consequence (right hand side of the rule) contains the set of possible states to which readings in this range correspond to. Assume a sensor s_1 and a set of a network's states $State = \{state_1, \ldots, state_m\}$ as defined by the previous

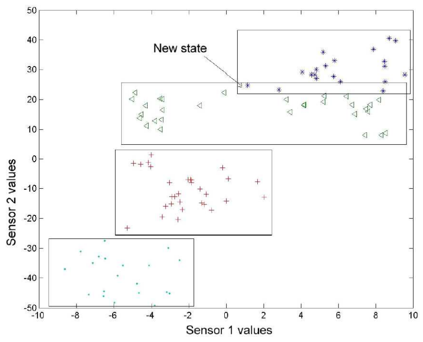


Figure 1. State Clustering: Projection of clusters (states) in the original space.

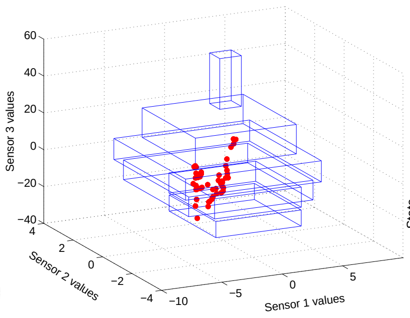


Figure 2. Example of state boundaries for 3 sensors and 6 states. 200 new system readings that are to be classified are shown as red filled dots.

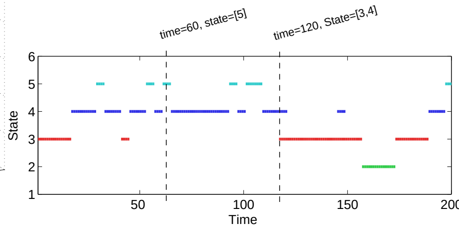


Figure 3. The 200 sensors readings from figure 2 are classified in the above 6 states.

clustering set. Let $\{v_1, \ldots, v_p\}$ be the set of intervals created by projecting the descriptions of the clusters into the domain of the readings of s_1. The rules for the network states will have the following form:

$$reading(s_j) \in v_p \rightarrow state(s_j)$$

where $reading(s_j)$ denotes the reading of the sensor s_j at a specific time point and $state(s_j)$ is a set $\{state_i, \ldots, state_k\} \subseteq State$. The meaning of the rule is that if the reading of the sensor s_j ($reading(s_j)$) lies in the interval $v(p)$, the network can only be in one of the states $state_i, \ldots, state_k$.

4.3 The Monitoring Process: Finding the state of a sensor network

As we have already mentioned in previous sections the readings of each sensor can only partially define the state of the network. Moreover, since the same readings of a single sensor can correspond to different states of the network, a sensor may not be able to define its state in relation to the overall system and thus it is considered to be in more than one state at a specific time point.

More specifically, the low level sensors send their state estimates to their parent sensors (according to the defined hierarchy). The parent sensors exploit the knowledge of their children and based on a voting approach define the current state of the respective subset of the network. Then the root (sink) makes the final decision for the network state taking the intersection of their children estimates for the network state. Once the state of the network has been defined the root of the network tree sends a message to all the sensors in the network informing them for the current state of the overall system.

Algorithm 1 summarizes the main steps of the procedure for computing the state of a sensor network based on the state estimates of its nodes.

Algorithm 1 Compute-network-state

```
Input: Set of sensors, $S = \{s_1, \ldots, s_n\}$
    States of the network, $State$.
    Sensor network hierarchy
Output: Network state.

1:  Definition of the sensors' state in the network tree
2:  if $s_i$ is leaf then
3:      $s_i$ propagates its state, $state(s_i)$, to its parents
4:  else
5:      $s_i$ receives the state of its children and computes
            $state(children(s_i)) = \cap_{s_k \text{ is child of } s_i} state(s_k)$
6:      computes $s_i$'s new state wrt the network state
            $state'(s_i) = state(s_i) \cap state(children(s_i))$
7:      if $state'(s_i) <> \{\}$ then
8:          propagate $state'(s_i)$
9:      else
10:         propagate $(state(s_i), s_i)$, and for each child $s_k$, also
                propagates $(state(s_k), s_k)$.
11: end if
12: The root of the network tree returns the state of the overall
        system.
```

In Figure 2 and Figure 3 we give an example of how our algorithm operates. For simplicity we assume only 3 sensors, and we plot 200 consecutive values of these sensors in 3-D. In Figure 2 we show a clustering of the values in 6 clusters, and show the descriptions of the clusters. Figure 3 depicts how the state of the network changes over time.

We note here that although the network DAG assists with the whole process of defining the network state, the state processing and clustering steps are independent of the actual tree organization.

Being outside defined states: It is possible that the system cannot converge to a specific state for the network. In other words, the intersection of all sensor state estimates is empty, and thus the root cannot find out the state of the network. Essentially, this means that the current network situation is not inside any of the regions that describe the defined states. This can happen for a set of reasons. A sensor may be faulty, or the network is seeing a situation not present in the current set of states. For example the nodes can fail due to energy depletion or their readings may drift due to environmental interference. This results in a wide variety of state estimates from the sensors and hence the system cannot define the state of the whole network based on the intersection of the sensors' states.

To handle this problem, we assume that when a node is not able to define the intersection of their children's state estimates (i.e. the intersection of their states is empty), it propagates the set of their children ids and their respective state estimates, instead of its intersection, to its own parent(s) as shown in Algorithm 1, Line 10. This procedure continues until all the state estimates reach the root.

It is important to note that although the root is not able to define the specific state of the network, it can *pinpoint* the region where the problem occurs. This is because the root can use this information to identify, based on the set of states that it receives, the range of the sensor readings in the whole system. Using this information it can detect the sensor or the set of sensors where the error has occurred.

We stress that our approach provides a mechanism for monitoring the system even in case of network faults and/or the presence of inadequate definitions of the network states: whenever the state definitions do not provide an advantage, our approach gracefully reverts to the case where no processing is done in the network and values are propagated to the root. In addition, our approach assists with pinpointing the problem and the subsystem where it occurs.

Evolution of the system: Our approach considers the intersection of the sensor states in order to define the state of the overall system, hence each sensor needs to propagate its state to its parents only if there is a change in relation to its previous state estimate. Then if a non-leaf sensor does not receive in a specific time period input from one of its children it uses the state that it has cached for this sensor. This reduces the number of messages sent among the sensors, improves resource savings, and minimizes the energy consumption and the workload in the network. However, one can argue that the above assumption can result in out of consensus state estimates since all the sensors do not change their state at the same time. Even if at a specific point in time it seems that there may not be a general consensus among the sensors, the system will reach a steady state when all new sensors states have been propagated to the sink.

5 Experimental Evaluation

In our experimental evaluation we focus mainly on evaluating the scalability and the robustness of our approach.

Our simulation testbed uses basic modules from the TAG [3] simulator (written in Java) in order to define the network topology and the sensors' communication. We implemented new modules for computing the network status.

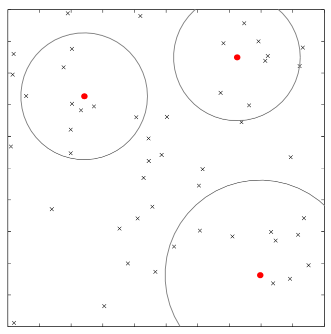
Our simulation scenario is the following. The nodes are randomly placed on a square grid, with the sink placed in the middle. The communication range was set to $\sqrt{2}$. For each run, the query is disseminated once from the sink at the beginning of the simulation. The desired number of states is also given as a parameter. To jumpstart the process each sensor transmits 10% of its readings, and we execute our clustering algorithm on this set of data. Once the states are found and the rules are transmited back to the sensors, we apply the online monitoring process on the remaining readings.

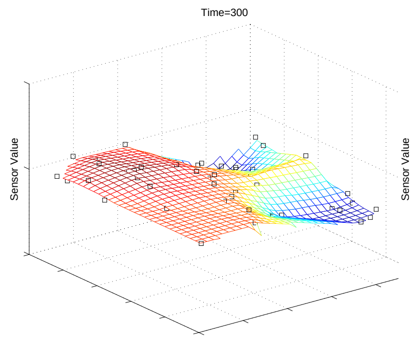
Datasets: The synthetic collection of sensor readings was created as follows. In order to generate datasets that resemble the characteristics of natural phenomena, the following approach was used. We assume a number of widely distributed points represented as dots in Figure 4a around which a set of sensors are distributed. These are the *points of influence* that affect the values of the sensors. On these points $2d$ Gaussian distributions are set. The readings of a sensor are a function of its distance from each point of influence: the circle drawn around the point defines a range at which its influence is diminished by half. As an example, the points of influence could designate the existence of an interesting event, e.g. some source emitting heat. Figures 4(b)-(c) present the distribution of the values of different sensors (represented by their location) at specific time points. The peaks and valleys in these figures are indicative of the different clusters into which the values of sensors can be organized, i.e. show the sensor states based on their readings. Unless stated otherwise, the default number of generated values for each sensor is set to 500.

To test our framework under a realistic application scenario we used weather data, which include temperature measurements from 32 stations from Washington and Oregon (http://www-k12.atmos.washington.edu/k12/grayskies). Each one of the 32 collections includes measurements on an hourly basis, for 208 days between June 2003 and June 2004 (i.e. has a length of 4990 values).

5.1 Experimental Results

Message transmission savings: The first set of experiments investigates the savings in message transmissions that our method introduces. The savings are expressed with re-





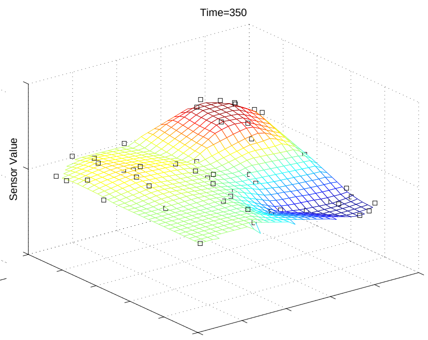


Figure 4. Data Generation: (a) Distribution of sensors in the network, (b)-(c) Sensors' readings at different time points: x, y give the location of the sensor, z gives the reading

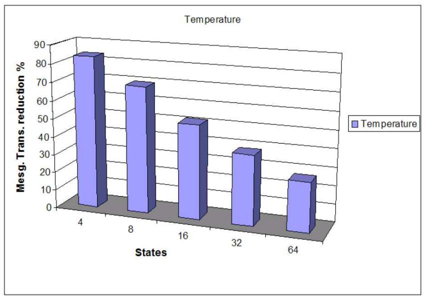


Figure 5. Message transmission reduction vs. the number of network states using real data.

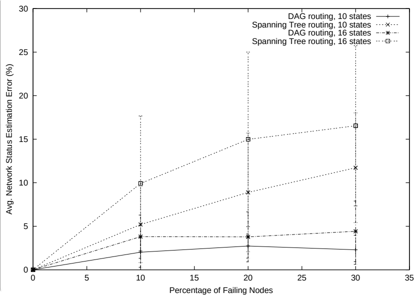


Figure 6. Network status estimation vs. the percentage of failing nodes

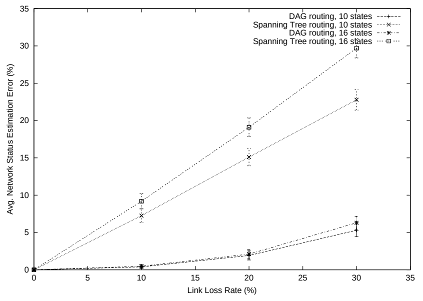


Figure 7. Network status estimation vs. the link loss rate

spect to the centralized strategy. We evaluate the performance of our approach experimenting with the temperature dataset (we obtained similar results with the synthetic datasets). For the experiments with the real datasets the number of sensors is set to 32, i.e. equal to the number of collections of temperature readings. Figure 5 shows the reduction of number of messages sent versus the number of states. We observe that there is significant reduction in the number of messages sent when the hierarchical structure of the network is used in order that the sensors propagate their state to their parents. Also, as expected, the number of messages increases as the number of states increases, since the more states we consider the higher the probability the readings of a sensor to be assigned in different states during a specific time period is.

Robustness wrt. failures: In another set of experiments we explore the robustness of our approach with respect to failures. We ran experiments introducing both standard link losses and node failures. The link losses affect the network operation momentarily and do not introduce major topology changes. Those changes are inflicted by node failures. In our simulation the node failures were modeled as follows. During a run of a simulation we set a certain percentage of nodes to fail, which yields a total number of nodes f that fail. These failures appear during the simulation uniformly at random, i.e. if D is the length of the simulation, at each time instant a failure may happen with probability $\frac{f}{D}$, and then a node is chosen randomly and is disabled.

We tested the routing scheme that builds a directed acyclic graph (DAG) on the network (i.e. nodes have multiple parents) and the one that uses a spanning tree, i.e. each node has a single parent. We attempt to measure the robustness of our approach by reporting the *average network status estimation error* (ANSEE), which is defined as follows. We repeat each run 100 times, since random failures (link losses and node failures) are introduced. We collect how many times, say f_i, during each repetition i of the experiment the estimated network status was different from the actual status of the network, as reported without introducing any failures. We report the performance values averaged over the 100 runs of each experiment.

Figures 6 and 7 show the ANSEE when node failures and link losses were introduced, respectively. The results that we report here were obtained using synthetic datasets on a sensornet of 200 nodes, while monitoring 10 and 16 states. Similar trends were observed using different configurations (number of nodes and states), but we omit their presentation due to space limitations. We observe that when we fix the number of network states, the routing using a DAG always yields significantly less error than the one using a spanning tree, both for node failures and link losses. On the other hand, there is no extra cost in terms of exchanged messages, since a message going upstream toward the sink is broadcast to the desired recipients (i.e. parents). The results in Figures 6 and 7 show that for a given routing scheme, monitoring more states introduces more uncertainty and error in the estimation, at the presence of communication failures. This happens because as the number of states increases, the mapping of each sensor's readings to the states is more volatile, i.e. there are more frequent changes in the state that the sensor reports. This is the cause for the need to report the change of state more often. Thus, when failures occur, the final network estimation is expected to include greater uncertainty, which is expressed as our ANSEE metric.

Finally, the introduction of permanent node failures do not severely adverse the network status computation. The communication paths are broken momentarily and, eventually, are re-established. The nodes not having heard of their parent(s) for a number of epochs, enter a *parent search* mode. The issues involved here are orthogonal to our approach, since they relate to the routing scheme used. On the other hand, link losses that may appear with the same rate at each attempt for a transmission, inflict higher ANSEE, especially when a spanning-tree routing scheme is employed.

6 Conclusions

In this paper, we address this problem proposing an approach that determines the state of the network by combining the partial knowledge of the individual sensor nodes. It uses a learning phase that clusters the sensors' readings on-line in order to define the states of the network. The results of the clustering are used to define a mapping from the multidimensional space of sensor readings to a set of states. Using this mapping our approach provides efficient cooperation of the sensors through an hierarchical organization of the network to define the overall status of the system. Our experimental results show that our approach enables efficient communication among the sensors, minimizing the number of the exchanged messages while it also allows fault tolerance through a multi-path communication among the sensors.

Acknowledgments.We would like to thank Samuel Madden for providing us with the source code of the TAG simulator. The work of Dr. M.Halkidi is funded by EU Commission(MOIF, 509920). The work of Dr. Kalogeraki and Dr. Gunopulos is funded by NSF.

References

[1] W. Zhang and G. Cao, "Optimizing tree reconfiguration for mobile target tracking in sensor networks." in *Proc. of IEEE INFOCOM*, Hong Kong, March 2004.

[2] J. Aslam, Z. Butter, F. Constantin, V. Crespi, G. Cybenko, and D. Rus., "Tracking a moving object with a binary sensor network." in *Proc. of SenSys Conference*, Nov. 2003.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *Proc. of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[4] Z. B. Tang, K. R. Pattipati, and D. L. Kleinman., "Optimization of distributed detection networks: Part ii generalized tree structures," *IEE Trans. Syst. MAn Cybern.*, vol. 23, pp. 211–221, 1993.

[5] S. Alhakeem and P. Varshney, "Decentralized bayesian hypothesis testing with feedback." *IEE Trans. Syst. MAn Cybern.*, vol. 26, pp. 503–513, 1996.

[6] P. F. Swaszek and P. Willett, "Parley as an approach to distributed detection," *IEE Trans. Aerospace Elect. Syst.*, vol. 31, pp. 447–457, 1995.

[7] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in networks," *SIGMOD Record*, vol. 31, no. 3, 2002.

[8] S. Madden, W. Hong, J. M. Hellerstein, and M. Franklin, "Tinydb web page. http://telegpah.cs.berkeley.edu/tinydb."

[9] C. Hsin and M. Liu, "A distributed monitoring mechanism for wireless sensor networks," in *ACM Workshop on Wireless Security (WiSe)*, 2002.

[10] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Erstin, "A wireless sensor network for structural monitoring," in *SenSys*, November 2004.

[11] J. Considine, F. Li, G.Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *IEEE ICDE*, 2004, pp. 449–460.

[12] E. Xing, A. Ng, M. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-inforformation." in *Proc. of Advances in Neural Information Processing Systems. (NIPS)*, 2002.

[13] S. Basu, M. Bilenko, and R. Mooney, "A probabilistic framework for semi-supervised clustering," in *ACM SIGKDD*, 2004, pp. 59–68.

[14] M. Halkidi, D. Gunopulos, N. Kumar, M. Vazirgiannis, and C. Domeniconi, "A Framework for Semi-Supervised Learning based on Subjective and Objective Clustering Criteria," in *IEEE ICDM*, 2005.

[15] M. Halkidi and M. Vazirgiannis, "Clustering validity assessment: Finding the optimal partitioning of a data set," in *ICDM*, 2001.

[16] S. Guha, A. Meyerson, N. Mishra, and R. Motwani, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, 2003.

[17] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient processing of spatial joins using r-trees," in *Proc. of ACM SIGMOD Conference.*, 1993.