Unsupervised Outlier Detection in Sensor Networks Using Aggregation Tree[*]

Kejia Zhang, Shengfei Shi, Hong Gao, and Jianzhong Li

Database Research Center
School of Computer Science & Technology
Harbin Institute of Technology, Harbin, China
`veron1967@163.com`

Abstract. In the applications of sensor networks, outlier detection has attracted more and more attention. The identification of outliers can be used to filter false data, find faulty nodes and discover interesting events. A few papers have been published for this issue. However some of them consume too much communication, some of them need user to pre-set correct thresholds, some of them generate approximate results rather than exact ones. In this paper, a new unsupervised approach is proposed to detect global top n outliers in the network. This approach can be used to answer both snapshot queries and continuous queries. Two novel concepts, *modifier set* and *candidate set* for the global outliers, are defined in the paper. Also a *commit-disseminate-verify* mechanism for outlier detection in aggregation tree is provided. Using this mechanism and the these two concepts, the global top n outliers can be detected through exchanging short messages in the whole tree. Theoretically, we prove that the results generated by our approach are exact. The experimental results show that our approach is the most communication-efficient one compared with other existing methods. Moreover, our approach does not need any pre-specified threshold. It can be easily extended to multi-dimensional data, and is suitable for detecting outliers of various definitions.

1 Introduction

Wireless Sensor Network (WSN) consists of hundreds or thousands of small, resource limited sensor nodes except one or more powerful nodes gathering the information of others [3,4]. We call the powerful node the sink or the base station (BS). Since sensor nodes (except the sink) are battery-powered and it is usually impractical to install new batteries, the WSN designers must find ways

[*] Supported by the Key National Natural Science Foundation of China under Grant No. 60533110; National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303000; the Key National Natural Science Foundation of Heilongjiang Province; the National Natural Science Foundation of China under Grant No. 60473075; Program for New Century Excellent Talents in University "NCET" under Grant No. NCET-05-0333.

to save energy and prolong the living time of WSN. The researches [1,2] reflect that for a sensor node, the communication cost is often several orders of magnitude higher than the computation cost. Therefore, minimizing communication overhead becomes the most important designing goal.

Generally, given a dataset S, an outlier is a data deviating markedly from other members of S. In a dataset, we can calculate *deviation rank* for each data to indicate its rank of deviating from others. Using different measurement criteria, data's deviation rank has different meanings. For example, for data x, we can use the *distance to kth nearest neighbors*[1] to measure its deviation rank [6], and we can also use the inverse of the number of neighbors within distance α, namely the inverse of $N(x, \alpha)$, to measure x's deviation rank [7]. Now the definition of outlier is: given a dataset, an outlier is a data whose deviation rank is larger than a given threshold. We can see that to define an outlier, a given deviation rank measurement criterion and a pre-specified threshold are needed.

In sensor networks, it is more useful to detect global outliers. Assume there are m sensor nodes in the network, and datasets $S_1, \ldots, S_m$ are the sample sets of these nodes. A global outlier is an outlier in the global set $\bigcup_{1 \leq i \leq m} S_i$. Compared with the data produced by sensors in the entire history, users are more interested in the data produced in a sliding window W of size $|W|$, especially in the current sliding window. Assuming the current time is t, the current sliding window can be seen a time interval from $t - |W|$ to t. Therefore, in a sensor network, $S_1, \ldots, S_m$ can be seen the datasets sampled by sensor nodes in current sliding window. Besides, since it is too difficult to set a correct threshold to define outliers while we don't know the distribution of data, our approach detects global top n outliers with largest deviation rank in global range, and just needs users to give the criterion to measure data's deviation rank.

Several characteristics of WSNs make the identification of outliers useful for analysis. The limited resource and capability make the data generated by sensor nodes undependable and inaccurate. Especially with power exhausting, the probability of generating erroneous data will grow rapidly. Therefore, we need a method to filter the data much different from others, i.e., outliers. Besides, we can also identify the faulty nodes by finding the nodes always generating outliers. Even if we are certain of the quality of the readings reported by sensor nodes, detection of outliers provides an efficient way to focus on the interesting events. For example, to supervise the process of manufacture, a machine is fitted with sensors monitoring its operation. When we find that the readings of temperature or vibration amplitude generated by a sensor are always much different from the data sampled by other nodes, there must be something wrong in the part this sensor monitoring.

To this day, we have already found a few papers [8,9,10,11,14,15,16] studying the problem of outlier detection in sensor networks, but works are still far from enough. [9,10] detect global outliers whose deviation rank in global range is larger than a pre-specified threshold. They need user to pre-set a correct threshold to find outliers. They use *kernel density estimator* to estimate the distribution of

[1] Here "neighbors" means the data around a given data point.

sensor readings, and use the probability density function to estimate $N(x, \alpha)$ for each data point x. Their approaches generate approximate results, which may filter many interesting events with small probability of occurrence. Even if we can tolerate the approximate results, we can not give a correct threshold to define outliers while we don't know the distribution of all the original sensor readings. [11] intends to clean outliers for better analysis. It uses *wavelet approximation* to clean local sudden-burst outliers in every node, and uses *dynamic time wrapping* to detect and remove outliers of small range (2 hops) which last for a long period. If we use this approach to detect outliers, it can only give an approximate result of outliers in small area, but not the exact global result in the whole network. [8] is similar to our approach. By using the definitions of outlier's *support set* and exchanging messages among all the nodes, it can exactly detect global top n outliers. It has two main disadvantages. One is that since it does not adopt any structure and every node uses broadcast to communicate, there is too much communication wasted while exchanging global information among all the nodes. The other is that its algorithm will end when all the nodes stop transmitting message, so it can not give a definite ending condition which can be judged by the sink. In other articles, [14] revises outliers using EM algorithm; [15] detects spatio-temporal outliers in stable distribution; [16] constructs a Bayesian Belief Network over WSN, and detects outliers by computing the likelihood of tuples.

In this paper, we propose a communication efficient approach to detect global top n outliers. We do not need any pre-specified threshold except the criterion (function) to measure outlier's deviation rank. Our approach is suitable to answer both snapshot and continuous queries. The basic idea is to build all the nodes in a structure of aggregation tree. Then let every node in the tree transmit some useful data to its parent after gathering all the messages from its children. The sink uses the information from its children to calculate global top n outliers, and floods these outliers for verification. If any node finds that it has two kinds of data which may modify the global result, it will send them to its parent in an appropriate time interval. All these processes are done level-by-level in the tree. Through a few such *commit-disseminate-verify* processes, all the nodes will agree on the global result calculated by the sink. The contributions of this paper are:

- We propose a communication efficient unsupervised approach for outlier detection in sensor networks. In our approach, we adopt the structure of aggregation tree. And we design a *commit-disseminate-verify* mechanism for outlier detection using aggregation tree. This mechanism can exactly detect outliers through exchanging short messages in the whole tree.
- We develop the definition of outliers support set and define two kinds of data which may affect the result of global outliers: *modifier set* and *candidate set* for the global outliers. By only transmitting these two kinds of data to upper node, every node can save a lot of communication. And the messages in the network are very short.
- We prove the correctness of our approach theoretically, i.e., the outliers reported by our algorithm are exact global top n outliers.

2 Preliminary

In this section, we introduce some necessary background definitions and notations.

The definition of data's deviation rank can be seen in section 1. For a data x in dataset S, we use $R(x, S)$ to denote the deviation rank of x in S. Here R is a deviation rank function. For different measurement criteria, $R(x, S)$ equals different values. We suppose in all cases, R satisfies the following theorem [8]:

Theorem 1. *For given $S_1 \subseteq S_2, R(x, S_1) \geq R(x, S_2)$.*

This theorem implies that the data belonging to a larger dataset has fewer opportunities to become an outlier.

We use the expression $\mathscr{A}[S]$ to denote the top n outliers in dataset S detected with algorithm $\mathscr{A}$. In $\mathscr{A}$, the deviation rank function is R. So we have:

$$\mathscr{A}[S] = \{x_1, \ldots, x_n \in S \mid \forall 1 \leq i \leq n \ \ and \ \ y \in S \setminus \mathscr{A}[S], \ \ R(y, S) \leq R(x_i, S)\}$$

Definition 1 (Support set). *Given a set $S_0 \subseteq S$ and a deviation rank function R, S_0 is called the* support set of $x \in S$ over S *if $R(x, S_0) = R(x, S)$ [8]. In all the support sets of x over S, we choose one with smallest size as the* smallest support set *of x over S, and use the expression $[S|x]$ to denote it. For given $S' \subseteq S$, we write $[S|S']$ to denote $\bigcup_{x \in S'} [S|x]$.*

In a distributed system, there are sets $S_1, \ldots, S_m$. We want to detect global top n outliers, i.e., $\mathscr{A}[\bigcup_{1 \leq i \leq m} S_i]$. Assume we have found global top n outliers and their deviation rank using a subset of $\bigcup_{1 \leq i \leq m} S_i$. Both the outliers and their deviation rank may be not right for not using the whole set. $A_g = \{x_1, \ldots, x_n\}$ denotes current global outliers. $R_g = \{R(x_1), \ldots, R(x_n)\}$ denotes their current deviation rank in global range.

Definition 2 (Modifier set). *For set S_i, its* modifier set *for global outliers consists of the data whose distance to a global outlier is less than the global outlier's deviation rank. We use $M(S_i)$ to denote it, and*

$$M(S_i) = \{x | x \in S_i, \exists current\ global\ outlier\ x_g \in A_g, \ \ |x - x_g| < R(x_g)\}$$

Here we assume R is a function of distance. If using the inverse of $N(x, \alpha)$ to measure data's deviation rank, α should be written instead of $R(x_g)$ in the formula.

Definition 3 (Candidate set). *For set S_i, its* candidate set *for global outliers consists of the data which may be candidates of correct global outliers, i.e., the data whose local deviation rank is larger than a global outlier's. We use $C(S_i)$ to denote it:*

$$C(S_i) = \{x | x \in S_i, \exists current\ global\ outlier\ x_g \in A_g, \ \ R(x, S_i) > R(x_g)\}$$

3 Unsupervised Outlier Detection Using Aggregation Tree

In this section, we describe our approach for efficiently detecting global top n outliers. In our algorithm, we assume there is a topological tree rooted at the sink. There are various methods [5] for constructing the aggregation tree according to different application requirements. And we do not rely on a specific tree construction algorithm as long as there is one.

For clearness, we first describe a simple aggregation tree construction algorithm. At the beginning, the sink broadcasts a beaconing message including its own ID and its depth 0. When a node, say x, receives a broadcast message at its first time from a node y, x assigns its depth to be the depth of y plus one, and its parent to be y. Then, x broadcasts the message recursively. This process continues until all the nodes have received beaconing message. For energy efficiency, we can join the outlier detection query into the beaconing message. When we disseminate a new query, the aggregation tree is constructed.

3.1 Algorithm for Snapshot Query

Snapshot query is a kind of queries which collect data about now or some other time point. In our model, snapshot query queries once about the global top n global outliers in current sliding window. Through the dissemination of query, an aggregation tree is constructed. To answer this kind of queries, our algorithm uses the *commit-disseminate-verify* mechanism. For a node N_i in the tree, we use S_i to denote the sample set of N_i in current sliding window. We define the *knowledge* of N_i as $\overline{S_i} = S_i \cup \{\text{data received from } N_i\text{'s children}\}$. And we use dataset HS_i to record the data N_i has sent to its parent.

Initially, every node commits for the first time. Through the first committing, we hope that every node N_i in the tree has a coarse view about the data generated in the sub-tree rooted at N_i. And we also hope not to consume too much communication. Therefore, every node must transmit the most useful dataset for outlier detection to its parent. For node N_i, it sends its local outliers $A_i = \mathscr{A}[\overline{S_i}]$ and their smallest support set $SA_i = [\overline{S_i}|A_i]$ to its parent after incorporating all the data from its children into its $\overline{S_i}$. Meanwhile, N_i sets $HS_i = A_i \cup SA_i$. At the end of this process, the sink calculates the global top n outliers for the first time which may be not right. We do this because though the outliers of parent (the top n outliers in the dataset of the sub-tree rooted at the parent) may not be outliers of its children, they come from the union of these children's outliers with high probability, especially when the data from different nodes is nearly the same.

To disseminate, the sink floods the unverified global outliers and their deviation rank. After receiving that, every node verifies and commits again. For node N_i, it calculates its $M(\overline{S_i})$, sets $Info_i^1 = M(\overline{S_i}) \backslash HS_i$, and waits for enough time to receive messages from all its children. After incorporating all the data from its children into its $\overline{S_i}$, N_i calculates its $C(\overline{S_i})$, and sets

$Info_i^2 = C(\overline{S_i}) \cup [\overline{S_i}|C(\overline{S_i})] \setminus HS_i$. If there is remaining data in $Info_i^1$ and $Info_i^2$, N_i sends $(Info_i^1, Info_i^2)$ to parent node and adds them into HS_i. Why we append the smallest support set of $C(\overline{S_i})$ to $Info_i^2$? The reason is that upper node can re-calculate deviation rank of the data in $C(\overline{S_i})$ exactly after incorporating $Info_i^2$ into a larger dataset.

After gathering all the committed information from its children, the sink recalculates global top n outliers. If the result is different from last one (either the outliers or their deviation rank), the sink will disseminate again until the result does not change.

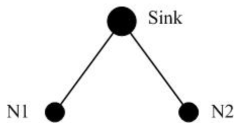

Fig. 1. A simple example. $S_1 = \{0.7, 1.3, 1.6, 2.0, 2.4, 2.8, 3.2, 8.6\}$. $S_2 = \{4.5, 8.2, 9.1, 9.3, 9.7, 10.2, 11.6, 12.1\}$. $S(sink) = \{3.0, 5.9, 6.4, 7.1, 7.9, 10.8, 11.3, 13.0\}$.

Next, we give a simplest example to demonstrate our algorithm. As shown in figure 1, the sink only has two children N_1 and N_2. We use *distance to the nearest neighbor* to measure outliers' deviation rank. The number of outliers we want to detect is $n = 1$. At the beginning, the sample set of N_1 N_2 and the sink are S_1 S_2 and $S(sink)$, as shown in the caption of the figure. We can see that the global outlier is 4.5.

To commit for the first time, N_1 calculates its local outlier $A_1 = \{8.6\}$ and A_1's smallest support set $SA_1 = \{3.2\}$, sends $A_1 \cup SA_1$ to the sink, and sets $HS_1 = \{3.2, 8.6\}$. Similarly, N_2 sends $\{4.5, 8.2\}$ to the sink and records them. After receiving, the knowledge of the sink becomes $\{3.0, 3.2, 4.5, 5.9, 6.4, 7.1, 7.9, 8.2, 8.6, 10.8, 11.3, 13.0\}$. Then, the sink finds that the global outlier is 13.0 and its deviation rank is 1.7 (distance to 11.3), which are not verified.

To disseminate, the sink broadcasts the binary group $(13.0, 1.7)$. After receiving that, N_1 and N_2 verifies and commits again. N_2 checks if it has some data whose distance to 13.0 is less than 1.7, and sets its $M(\overline{S_2}) = \{11.6, 12.1\}$, $Info_2^1 = M(\overline{S_2}) \setminus HS_2 = M(\overline{S_2})$. Then N_2 checks if it has some data whose local deviation rank is larger than 1.7, and sets its $C(\overline{S_2}) = \{4.5\}$, $Info_2^2 = C(\overline{S_2}) \cup [\overline{S_2}|C(\overline{S_2})] \setminus HS_2 = \{4.5, 8.2\} \setminus \{4.5, 8.2\} = \emptyset$. Finally, N_2 sends $(\{11.6, 12.1\}, \emptyset)$ to the sink. Meanwhile, N_1 finds that its $Info_1^1 = \emptyset$ and $Info_1^2 = \emptyset$, and sends nothing to the sink. After receiving messages from its children, the knowledge of the sink becomes $\{3.0, 3.2, 4.5, 5.9, 6.4, 7.1, 7.9, 8.2, 8.6, 10.8, 11.3, 11.6, 12.1, 13.0\}$. Then the sink finds that the global outlier has become 4.5 with deviation rank 1.3, so it disseminates again and floods $(4.5, 1.3)$.

Receiving the information about the new outlier, both N_1 and N_2 find that they have nothing to send back to the sink. So the sink will affirm that the global outlier is really 4.5 with deviation rank 1.3. N_1 and N_2 can know that too.

3.2 Update of Outliers for Continuous Query

Continuous query is a kind of queries which collect data for a long period. In our model, continuous query queries continuously about the global top n outliers in current sliding window for a long period. As time advances and the current sliding window moves forward, the sample set of every node changes continuously, so the result of global outliers must change in time. It is easy to modify the above algorithm to answer continuous queries.

Algorithm 1. Find_Outliers (N_i)

```
Input of $N_i$: outlier detection algorithm $\mathscr{A}$ and window size $\tau$
Output of sink node : global top $n$ outliers
Initially:
1. Every leaf node $N_i$ calculate its $A_i \leftarrow \mathscr{A}[\overline{S_i}]$ and $SA_i \leftarrow [\overline{S_i}|A_i]$, then sends $A_i \cup SA_i$
   to parent.
2. After incorporating all the data from its children into its $\overline{S_i}$, the intermediate node
   calculates its own $A_i$ and $SA_i$, then sends them to parent node.
3. After receiving from all its children, the sink calculates global outliers $A_g = \{x_1, \ldots,$
   $x_n\}$ and their deviation rank $R_g = \{R(x_1), \ldots, R(x_n)\}$, then floods $(A_g, R_g)$.
Case receive new global outliers' information or there is any change in $\overline{S_i}$:
4. Update the information about $A_g$ and $R_g$
5. Retire the data older than $\tau$ in $\overline{S_i}$
6. $Info_i^1 \leftarrow M(\overline{S_i}) \setminus HS_i$
7. IF $N_i$ is leaf node:
8.      $Info_i^2 \leftarrow C(\overline{S_i}) \cup [\overline{S_i}|C(\overline{S_i})] \setminus HS_i$
9.      $Message \leftarrow (Info_i^1, Info_i^2)$ and send $Message$ to parent
Case receive $Message = (Info_j^1, Info_j^2)$ from child $N_j$:
10. $\overline{S_i} \leftarrow \overline{S_i} \cup Info_j^1 \cup Info_j^2$
11. $Info_i^1 \leftarrow Info_i^1 \cup Info_j^1$
Wait for enough time to receive data from every child, then:
12. IF $N_i$ is not the sink:
13.      $Info_i^2 \leftarrow C(\overline{S_i}) \cup [\overline{S_i}|C(\overline{S_i})] \setminus HS_i$
14.      $Message \leftarrow (Info_i^1, Info_i^2)$ and send $Message$ to parent
15. ELSE                                   //$N_i$ is the sink node
16.      $A_g \leftarrow \mathscr{A}[\overline{S_i}]$      $R_g \leftarrow \{R(x, \overline{S_i})|x \in A_g\}$
17.      IF there is any change in $A_g$ and $R_g$:
18.           Flood $(A_g, R_g)$ to every node.
19.      ELSE
20.           Report $A_g$ as global top $n$ outliers
```

The algorithm designed for both snapshot query and continuous query is given in algorithm 1. The inputs of every node are the deviation rank function R, the number of reported outliers n and the sliding window size τ. R and n can be contained in algorithm $\mathscr{A}$, which we use to detect local outliers in every node. In algorithm 1, A_g denotes global top n outliers detected by the sink which may be not right, and R_g denotes these outliers' deviation rank calculated by the sink. Node N_i has child N_j.

Initially, every node commits for the first time (lines 1-3). If the sink finds new result of global outliers, it disseminates (lines 16-18). Otherwise, the sink reports global top n outliers (lines 19, 20). Receiving the information of new global outliers, every node verifies and assigns its $Info_i^1$ (lines 4, 6). After waiting for enough time to receive data from its children, every node assigns its $Info_i^2$ and commits (lines 7, 8, 12-14). As time advances, if there is any change in $\overline{S_i}$, N_i verifies by checking whether there is new data added into its $Info_i^1$ and $Info_i^2$ (lines 6, 13, 14). If there is useful data to send, N_i will transmit them in an appropriate time interval.

For further optimizing, as time advances, we could let every node delay for a fixed time to accumulate more changes before verifying and committing. Of course, this will cause the result of global outliers to defer to a certain extent. However, the exactness of the result can not be suppressed much in case that the sample set of every node changes smoothly. Even if we can not tolerate the delay of results and we need to detect outliers exactly in time, our experiments show that our approach performs much better than other existing methods.

4 Proof of Correctness

Our algorithm intends to find the exact global top n outliers. As described in the following two theorems, we can prove the correctness of our algorithm in two steps. Below, we use US to denote the union of all the original data in current sliding window, i.e., $US = \bigcup_i S_i$. And $\overline{S}$ denotes the final knowledge of the sink.

Theorem 2. *If the network setting and every node's sample set do not change, the algorithm will definitely terminate. When the algorithm terminates, for each global outlier x in A_g, its deviation rank calculated by the sink, i.e., $R(x)$, is the correct deviation rank in global range.*

Proof. Since network setting and every node's sample do not change, line 20 in algorithm 1 can make sure that the algorithm will definitely terminate. Lines 6 and 11 assure that for each outlier x in A_g, any data point closer to x than its current deviation rank $R(x)$ must be added into node's $Info_i^1$ and sent up until to the sink. When the algorithm terminates, at least one support set of x will be received by the sink (Otherwise, there must be some data closer to x than $R(x)$ not received by the sink). Therefore, when the algorithm terminates, $R(x)$ equals the deviation rank calculated using all the original sensor readings, i.e., $R(x) = R(x, US)$.

Theorem 3. *If the network setting and every node's sample set do not change, when the algorithm terminates, the outliers reported by the sink, i.e., A_g, must be equal to the correct global top n outliers $\mathscr{A}[US]$.*

Proof. Suppose to the contrary that there is a data x belonging to $\mathscr{A}[US]$ but not in A_g. Therefore, for at least one outlier y in A_g, there must be $R(x, US) > R(y, US)$. According to theorem 2, $R(y, US) = R(y)$, so we have $R(x, US) > R(y)$.

a) If x is in $\overline{S}$, according to theorem 1 and $\overline{S} \subseteq US$, we know that $R(x) = R(x, \overline{S}) \geq R(x, US)$. Then we can conclude that $R(x) > R(y)$, which means x should be in instead of y in A_g. **b**) If x is in the knowledge of node N_i(i.e., $\overline{S_i}$) but not in $\overline{S}$. Since $\overline{S_i} \subseteq US$, we have $R(x, \overline{S_i}) \geq R(x, US) > R(y)$ according to theorem 1. Lines 8, 9, 13 and 14 in algorithm 1 assure that any such data x must be added into N_i's $Info_i^2$, and sent to upper node until to the sink. Basing on the proof **a** and **b**, we can conclude that the existing of such data x is impossible by all means, and $A_g = \mathscr{A}[US]$.

The proof of correctness is under such a hypothesis: for every node in the network, its sample set does not change. That does not imply the algorithm is incorrect while answering the continuous query. As long as we can make sure that there is no change in every node's sample set when the algorithm is executing, i.e., the calculation of outliers is faster than the updating of data, the algorithm's results are certainly correct at any moment.

5 Experimental Evaluation

5.1 Experimentation Setup

We build a simulator implemented with 1000 lines of C++ code to evaluate the performance of our method. And we collect two kinds of data to measure the efficiency of approaches (y-coordinate), they are: (a) Average total number of packets received per node. (b) Average total number of data points/bytes received per node. Because the receiving energy consumption is the dominate term in energy use [13], we believe that these two kinds of data can give a good estimation of energy consumption.

We compared our approach with two approaches: centralized approach and the approach proposed by [8]. In centralized approach, all the nodes in the network periodically send their original data to the sink, and the sink detects outliers using its local algorithm.

The dataset we use comes from [12], which is a real-world dataset collected by Berkeley research lab. Because our algorithm has good scalability of data dimensionality (generally, data dimensionality does not affect the exactness and efficiency of our algorithm), we only use one attribute (temperature) to compose data point and measure the performance. One data points include following features: (1) ID of the node producing the data point. (2) The time stamp when the data point is generated. (3) The data value (temperature). Furthermore, for the reason that the environment in Berkeley's lab is very steady and the data from [12] is too smooth to reflect some abnormal events happening, we randomly insert some synthetic data with abnormal values into the dataset. Our algorithm detects outliers measured by *distance to k-nearest neighbor*. In our experiments, we set $k = 4$.

In the simulated experimentation, 50 nodes are deployed in a $250m \times 250m$ area. The node's transmission range is assigned to be 40m and each node has 4˜6 neighbors. Unlike in actual environment, we assume that the communication in the network is reliable and the transmission failure rate is zero. All the experiments run 1000 seconds of simulation time. As shown in the following graph, we evaluate the performance for different parameters (x-coordinate) of a) the size of sliding window. b) the number of reported outliers n.

In these figures, *Centralized* denotes the naive centralized approach gathering sensor readings from all the nodes and calculating the outliers in the sink. *IN-Broadcast* denotes the in-network outlier detection approach proposed by [8]. *NS-Tree* denotes the unsupervised approach using our algorithm. *NS-Delay-x* denotes the improved version of our approach. In this approach, each leaf node delays x seconds to accumulate more changes before reporting to upper node rather than updates the knowledge and reports to upper node immediately. The idea of this approach can be seen in section 3.2.

5.2 Impact of Sliding Window Size

As shown in figure 2, NS-Delay-5 is the most communication-efficient approach in the four, no matter how big the sliding window is. In NS-Delay-5, every leaf node delay 5 seconds to accumulate more changes before reporting to upper node. Even if we can not tolerate the 5 seconds delay of the result, the approach NS-Tree without any delay performs much better than the other two. With the size of sliding window growing, except the centralized approach, the communication overhead of the other three approaches decrease gradually. The reason for this phenomenon is that in larger dataset, more benefit can be achieved through analysis in-network and there are more chances to reduce communication.

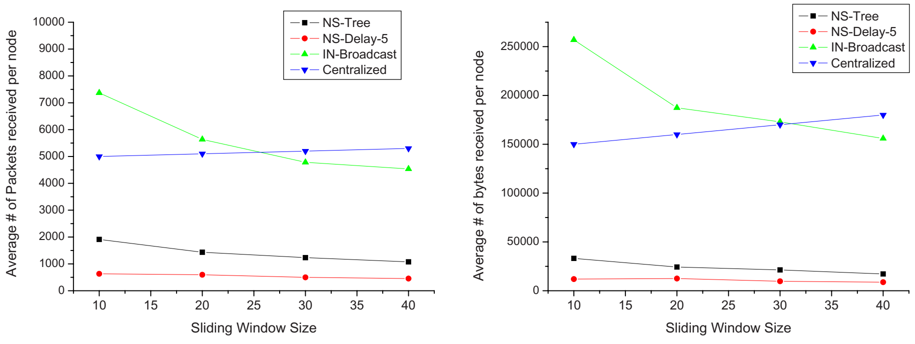

Fig. 2. Impact of sliding window size ($k = 4, n = 4$)

In figure 2, we find a problem that while the window size is smaller that 30, the IN-Broadcast approach performs even worse than the centralized method. Since each node in IN-Broadcast uses broadcast to communicate and the direction of transmission is not controlled by the algorithm, there is too much communication wasted in this approach. But IN-Broadcast has less average number of packets sent per node than the centralized method, because broadcast means that one node sends while many nodes receive.

5.3 Impact of Number of Reported Outliers

As shown in figure 3, NS-Tree and NS-Delay-5 still perform much better than the other two. As the number of reported outliers increases, the communication overhead of centralized approach does not change, because no matter how many outliers to detect, the sink still gathers data from all the nodes. On the other side, the communication overhead of the other three approaches increases with n. That is because more outliers we want to detect, more communication we consume. Like in figure 2, when n is larger than 4, IN-Broadcast performs even worse than the naive centralized method. Since the in-network analysis in IN-Broadcast is complex and not efficient, this approach by in-network analysis is not as good as naive centralized method while reporting a number of outliers.

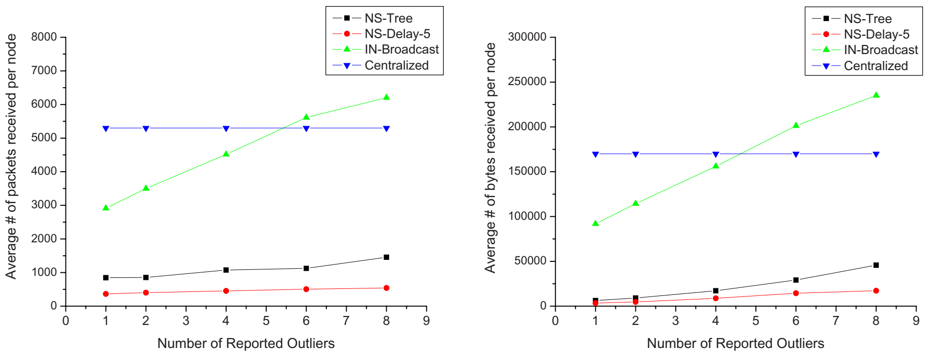

Fig. 3. Impact of number of reported outliers ($WindowSize = 40$, $k = 4$)

6 Summary

In this paper, we study the problem of outlier detection in sensor networks and propose a unsupervised approach to detect global top n outliers in a sliding window. Our approach is base on the model of aggregation tree and we design a *commit-disseminate-verify* mechanism for outlier detection in aggregation tree. Moreover, we develop the definition of outlier's support set and define two kinds of data which may affect the result of global outliers. By exchanging short messages in the whole tree, every node will get the information of correct global outliers. We proved the correctness of our algorithm strictly. Compared with existing approaches, our method is more communication-efficient and has advantages of unsupervised, result exactness, latency decrease and definite ending condition. The experiments confirm the efficiency of our method.

In the future, our works may be applying the idea of our algorithm in some other applications, designing an approximation model to decrease communication overhead further, or making node find some other useful knowledge through analysis in-network.

References

1. Shnayder, V., Hempstead, M., Chen, B.R., Allen, G.W., Welsh, M.: Simulating the Power Consumption of Large-scale Sensor Network Applications. In: SenSys (2004)
2. Gupta, P., Kumar, P.R.: The Capacity of Wireless Networks. IEEETrans. Information Theory 46(2), 388–404 (2000)
3. Warneke, B., Last, M., Liebowitz, B., Pister, K.: Smart Dust: Communicating with A Cubic-millimeter Computer. IEEE Computer Magazine, pp. 44–51 (January 2001)
4. Gunopulos, D., Kollios, G., Tsotras, J., Domeniconi, C.: Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. In: SIGMOD (2000)
5. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The Design of An Acquisitional Query Processor for Sensor Networks. In: SIGMOD (2003)
6. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient Algorithms for Mining Outliers from Large Datasets. In: SIGMOD (2000)
7. Knorr, E., Ng, R.: Algorithms for Mining Distance-Based Outliers in Large Datasets. In: VLDB, 24–27 (1998)
8. Branch, J., Szymanski, B., Giannella, C., Wolff, R.: In-Network Outlier Detection in Wireless Sensor Networks. In: ICDCS (2006)
9. Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., Gunopulos, D.: Online Outlier Detection in Sensor Data Using Non-Parametric Models. In: VLDB (2006)
10. Palpanas, T., Papadopoulos, D., Kalogeraki, V., Gunopulos, D.: Distributed Deviation Detection in Sensor Networks. ACM SIGMOD 32(4), 77–82 (2003)
11. Zhuang, Y., Chen, L.: In-network Outlier Cleaning for Data Collection in Sensor Networks. In: CleanDB (2006)
12. Intel Berkeley Research Lab. `http://db.lcs.mit.edu/labdata/labdata.html`
13. Crossbow Technology Inc. `http://www.xbow.com/`
14. Ash, J.N., Moses, R.L.: Outlier Compensation in Sensor Network Self-localization via the EM Algorithm. In: ICASSP (2005)
15. Jun, M.C., Jeong, H., Kuo, C.J.: Distributed Spatio-temporal Outlier Detection in Sensor Networks. In: SPIE (2005)
16. Janakiram, D., Reddy, A.M., Kumar, A.P.: Outlier Detection in Wireless Sensor Networks using Bayesian Belief Networks. In: Comsware (2006)