Comparative Analysis of Algorithms for Tree Structure Restoration in Sensor Network

Fan Bai, Ahmed Helmy
Department of Electrical Engineering
Univ. of Southern California
{fbai, helmy}@usc.edu

Abstract Sensor networks will be usually used for the collection of measured data. In many cases, a tree structure is formed for data query, data dissemination and other operations. As sensor nodes fail, this underlying tree structure is impacted or, at the worst case, disabled. In this paper, we propose and compare three algorithms to restore the tree structure for sensor network under network dynamics. Two of the algorithms use global information at the base station (or sink), while the third uses only local information. Through simulations, we observe that the localized algorithm outperforms the two global algorithms in terms of communication cost, corresponding energy and latency. In addition, we also gain a deeper insight into the performance tradeoff for algorithms in sensor network. We clearly identify energy-latency tradeoff for the global algorithms, as well as energy-accuracy (optimality) tradeoff for the localized algorithm.

I Introduction

With recent technical advances in micro-processor, memory and radio communication systems, large-scale sensor networks consisting of numerous cheap and small sensors are becoming more feasible. Recently, wireless sensor networks (WSN) have attracted plenty of attention from the research society, because of its potentials in both academia and industry.

One main problem in sensor network is to retrieve the measured data from sensors and deliver it at a sink or base station. To achieve this objective, a common practice is to establish an underlying structure for the sensor network linking all sensors and the base station. With the help of this underlying structure, ordinary operations like data query, data retrieval, data processing and aggregation, packet routing and even topology maintenance could be implemented efficiently. Some of the underlying structures may be implicit structures, such as GHT [4], the data-centric storage and routing infrastructure based on the hashing table mapping scheme. Other structures, such as tree structures (with the base station being the root of the tree) are explicitly set up, including Directed Diffusion [2] and GEM [1], among others. In this paper, we focus on and study the problem of efficiently maintaining a tree structure in the sensor network in the face of network dynamics and node failures.

In the past few years, numerous efforts are devoted to develop various novel energy-efficient protocols to prolong the lifetime of sensor network and delay the depletion of battery as late as possible. However, in reality, sensors are power-limited and error-prone, i.e., the sensor nodes susceptible to failure for various reasons. A sensor node may run out of power after many years of operations. Sensor nodes may crash because of software or hardware failures. Several sensor nodes in a specific region may be destroyed by natural disasters or malicious attacks. In brief, the failure of micro sensors in sensor network is unavoidable. Facing the inevitable sensor failure, the existing infrastructure of sensor network is subject to change, which in turn impacts or even disables, in the worst case, the normal operations of a sensor network. How to restore the infrastructure in the face of network dynamics is an important and challenging topic of research.

In this paper, we propose three algorithms for tree restoration in sensor networks. Two use global information and are called the global algorithms, and the third uses only local information and is called the localized algorithm. The two global algorithms include global flooding and the global circulating algorithm. In both algorithms, all the live sensors within the sensor field are directly or indirectly checked by the inquiry packets issued by the base station. Thus the tree structure could be re-established. Through simple practical analysis, we find that an energy-latency trade-off seems to exist for the global algorithm: global flooding algorithm incurs less latency but wastes too much energy on communication overhead, while global circulating algorithm generates less overhead but incurs excessive latency. Based on our analysis we argue that neither of these global algorithms is desirable for sensor network, where the energy is a scarce resource. Furthermore, we propose a localized tree restoration algorithm through localized, scoped, flooding. The basic idea of the localized flooding algorithm is to restrict the flooding only within the region where the sensor failure happens, thus, only the portion of the tree structure being affected by the failed sensors is repaired, while the other part of tree structure remains intact. The task of initiating the scoped flood is performed by an ancestor of the failed sensor.

Through simulation, the localized algorithm is shown to outperform the two global algorithms in terms of both latency and communication overhead. In contrast to the global algorithms, for this problem of tree restoration in the face of sensor failure, we believe that the localized algorithm provides an energy-efficient solution. Furthermore, through extensive simulations, we also realize that the performance of localized flooding algorithm (including energy-related metrics) is sensitive to the range of localized flooding (i.e., the K parameter). By varying this parameter, the various energy-accuracy/optimality trade-offs could be achieved in the localized algorithm.

We begin by discussing the related work in Section II. Then we describe the underlying tree structure used in our paper and introduce the terminology in Section III. In section IV, we describe the two global algorithms and conduct a simple theoretical comparison between their performance. The localized flooding algorithm is formally discussed in section V. The simulation result is illustrated in Section VI, and we also discuss the performance tradeoff related with issue of energy in this section. Finally, we conclude the paper in the Section VII.
II Related Work

In sensor network, tree structure is a commonly used underlying structure. In [3] a spanning tree is used to perform the node-to-base-station routing. In Directed Diffusion [2], the initial stage of interest dissemination and gradient setup is also based on the standard scheme for shortest-path tree setup. To deal with network dynamics, directed diffusion utilizes a soft-state timer that leads to repeated flooding with network dynamics; often expensive in large sensor networks. A recent paper [1] further extends the wide utility of underlying tree structure for sensor network: The tree-like GEM infrastructure is built to provide functionalities such as node-to-node routing, data-centric storage as well as data aggregation and data processing. Facing the node failure, a rather complex set of rules is applied to repair the broken tree structure.

The idea of restricted flooding is somewhat inspired by LAR [5], a MANET routing protocol which utilizes the location information to restrict the scope of route request at the phase of route discovery. In our localized flooding algorithm, we build upon the idea of controlled flooding. However, rather than relying on the explicit location information to determine the potential area for flooding, as shown in Section V, we only use the implicit ancestor-offspring relationship in the logical tree structure to limit the scope of flooding. We do not assume that location information is always available. Besides, the geographic-logical closeness exists in the tree-structure of dense sensor networks (see Section 5.1 for details).

Our work is closely related to the work in [6], which demonstrates an efficient base-station-dominant algorithm that could trace the identities of failed sensors with the moderate overhead. However, our crucial objective is not to determine the failed sensor, but also to reconstruct the underlying tree structure under network dynamics. In addition, in [6] the majority of work is done by base station because the long-range radio channel is optimistically assumed for base station, while in our paper the task of finding failed nodes and reconstructing the tree structure is achieved through the cooperation between sensors equipped with the normal radio.

III Sensor Network Model and Terminology

Within the region where the physical phenomenon happens, the measured data from various sensors is more or less correlated. And considering the power is a scarce resource, it is a natural thought to conduct the in-network data processing and data aggregation at the appropriate position of the tree structure: Each sensor sends its own data independently towards the base station along the tree structure. However, the upper-level node will aggregate the data packets from different lower-level nodes if their paths to base station are overlapping. Apparently, it forms a tree-like structure.

One obvious scenario in which tree-like structure could be applied is the sensor network used for the continuous monitoring, such as the perimeter defensive system, where each sensor is required to report its data to base station every T second. Even for the event-driven applications, the data flows from various sensors in the sensor field are able to aggregate along the routes back to base station. Hence, the aggregation tree structure is also a desirable underlying infrastructure for this kind of scenarios. In our opinion, because of the property of many-sensors-to-one-sink data flows and the strict requirement for power efficiency, tree structure is one suitable underlying infrastructure could be used in sensor network.

Therefore, in this paper, we assume the base station and tree-like structure model for sensor network. In this model, the sensors in the sensor field form a tree-like structure; the base station at the edge is the root of the tree. Each sensor has limited capacity like storage, processing capacity and power, so it only maintains the information about its parent and children of the tree structure or other direct neighbors. In this paper, we use the terminology node and sensor interchangeably. Compared to the normal sensors, the base station is relatively powerful.

![Tables maintained by each sensor](image)

Figure 1 (a) Flooding (b) Circulating (c) Localized Flooding

We say a node is hearable if the measured data from this node could be received by base station. A node is said as failed if this node runs out of power or it is destroyed by malicious attack, then this node is neither able to sense the nearby phenomenon nor to transmit the sensed data. A node is categorized as soundless if the measured data from this node could not be received by base station even though it is still alive. The region where set of all soundless nodes locates is called silent zone. The reason why base station cannot receive the data from silent zone is that the ancestor of soundless sensors in the tree structure is dead. The objective of the proposed algorithms is to restore the underlying tree structure of sensor network after some sensor fails, therefore, the soundless nodes could be reconnected to the repaired tree structure. Through the reestablished tree, the measurement data from soundless sensors could be received at base station again.

Whether a sensor fails or not is monitored by its direct neighbors within transmission range. In our scheme, each sensor is responsible to monitor the status of all its children within the tree structure. For each sensor, if one of the children ceases sending sensed data over time, it might send an error message back towards base station to report about the failure. For the
failed node, we call its parent as upstream detector. At the same time, the failure of sensor is also detected by its children in the tree structure, we call those children as downstream detectors. All the terms above are illustrated in Figure 2(c).

After clarifying the above terminology, we can describe the two global algorithms and one localized algorithm for tree reconstruction in the next two sections.

IV The Two Global Algorithms
In the face of sensor failure, the intuitive method to restore the underlying infrastructure of sensor network is to inquiry about the status of all nodes within the sensor network and reconstruct the whole tree structure from scratch. In this section, we describe two global algorithms and discuss their pros and cons.

4.1 Global Flooding Algorithm
The simplest algorithm is global flooding algorithm. In this algorithm, when a sensor is found as dead, the root initiates a flooding with the range of whole network. Thus, the new tree structure is reconstructed correspondingly.

Before we describe the details of global flooding algorithm, we need to introduce the table kept at each sensor used to establish and maintain the tree structure. The key information in the table is updated when the algorithms are applied. The table for global flooding algorithm is shown in Figure 1(a). ID field stores the globally unique id of the sensor, Parent field is to store the id of the parent for this sensor in the tree structure. Children_List field is to store the list of ids of all the children for this sensor in the tree structure, Level field is to record the level of this sensor in the tree structure, Sequence_Num field maintains the sequence number of global flooding initiated by the root. This number is to help sensors distinguish whether incoming FLOOD packet is a new one or not. Please note this number is only incremented and issued by root (base station) rather than any other sensors. In the global flooding algorithm, three types of packets are used: ERROR, FLOOD and SOLICIT packet. Their functionalities would be elaborated later.

Whether each sensor is failed or not is monitored by its neighbors in the way described in section III. Once certain upstream detector observes that some of its children fail, it sends an ERROR packet immediately towards the root to report the abnormality. After receiving the ERROR packet, the root increments the global flooding sequence number by one and it broadcasts a FLOOD packet (tagged with the new sequence number). When sensor receives a FLOOD packet, it will first check whether the sequence number tagged in FLOOD packet is a new one. If not, the sensor just discards the incoming FLOOD packet because it has been processed before. Otherwise, it updates its current Seq_Number field by the new sequence number in FLOOD packet, and revises the Parent field as the sender of incoming FLOOD packet. Afterward, a SOLICIT packet is sent back to its parent to establish the parent-children logical relationship. After the parent node updates its Children_list field, this SOLICIT packet is then forwarded towards root. As the last step, the sensor rebroadcasts the incoming FLOOD packet to all its neighbors. This procedure is repeated until all sensors receive the FLOOD packet.

In this way, the FLOOD packets can reach all the nodes alive in the sensor field. The nodes respond the FLOOD packet by sending its own information included in the SOLICIT packets, along the newly established tree structure. Based on the SOLICIT packets from all sensors, it is straightforward for root to construct the tree structure from scratch. In chorus, for each sensor, the necessary table entries to maintain the inherent tree structure including Parent field, Children_List field has also been set. In summary, for global flooding algorithm, after some sensors die, all sensors alive re-establish the new tree structure. This new tree structure may differ with the tree structure before node failure. The example of global flooding algorithm is shown in Figure 2(a). The detailed pseudo-code is provided in Appendix A.

4.2 Global Circulating Algorithm
The basic idea for global circulating algorithm is to visit all the nodes within the whole network and to rebuild the tree structure from scratch. Different with the fashion of inquiring the sensor field through multiple threads concurrently (as global flooding scheme), global circulating algorithm visits all the nodes in a one-by-one fashion by single inquiry process. Actually, the global circulating scheme is a modified version of DFS algorithm[8]: The CIRCULATING packet works like a mobile agent in sensor network, which traverses and establishes the new tree structure of the network in a DFS fashion.

We first describe the table kept at each sensor for the circulating algorithm. For each sensor, the table maintained is shown in Figure 1(b). Besides the fields discussed in the previous section, some new fields are added. Status field indicates the status of sensor, as specified by DFS algorithm: For each specific sequence number, White color represents that the node has not been visited by the CIRCULATING packet yet; Gray color represents that the node has been visited once but the visiting process is not finished yet (in other word, the CIRCULATING packet is currently visiting some sensor within the subtree rooted at this node); Black color represents that all the nodes in its subtree has already been visited. For every new global circulating sequence number, each sensor automatically sets its default value for Status field as white. In
addition, each node is responsible to notify its direct neighbors about the current status after its Status field has been changed. Only when the status of the node is gray (i.e., the CIRCULATING packet is currently at somewhere of its subtree), the Incoming field records the node from which the CIRCULATING packet comes, the Outgoing field records the node to which the CIRCULATING packet goes. These two fields are designed to facilitate the CIRCULATING packet to traverse the network. Only two types of packets are used in this algorithm: ERROR and CIRCULATING packet. In relation to the direction of movement with respective to the tree structure, as shown in Figure 2(b), the movement of CIRCULATING packet could be categorized into two types: downstream traversal indicates the movement of CIRCULATING packet from parent node to child node (i.e., go down in the tree), and upstream traversal represents the return of CIRCULATING packet from child node back to parent node (i.e., up in the tree). Then, we discuss the details of this algorithm.

When an ERROR packet is sent from upstream detector back to root, the root increments its circulating sequence number, which indicates the freshness of the CIRCULATING packet. Afterwards, root randomly chooses one of its direct neighbors whose status is still white as the beginning of circulating process. Then, it sets its Status field as gray and sends the single CIRCULATING packet to the chosen one.

Once a sensor with white status receives a CIRCULATING packet from root or high-level node with gray status, if the circulating sequence number has been processed before, this CIRCULATING packet is discarded. Otherwise, it updates its Parent and Level field as specified in the DFS algorithm, and then changes its Status field as gray. If none of its neighbors is still white, it then labels its Status field as black at once and returns the CIRCULATING packet back to its parent (that is an upstream traversal). Otherwise, if any of its neighbors whose status is still white for the new sequence number, it will keep its status as gray. Then, the CIRCULATING packet will be sent to the one of the unprocessed neighbors with white status (that is a downstream traversal), until none of the direct neighbors is still with white Status field.

By this way, the CIRCULATING packet keeps traversing downstream until it finally reaches a node with no white neighbors. At this moment, the CIRCULATING packet can hardly go further and this node (leaf node) labels its Status field from gray to black. Then, the CIRCULATING packet goes up to revisit the parent node from which it comes (upstream traversal). Once the CIRCULATING packet returns back to the parent node, if none of the neighbors of the parent node is still white, the parent node labels it as black and the CIRCULATING packet returns back to the parent of parent node (upstream traversal); if not, it will send the CIRCULATING packet to one unprocessed neighbor with white status (downstream traversal). The example is shown in Figure 2(b). The number for each node shown in this figure illustrates the order that the nodes have been visited by the global circulating algorithm.

Following the fashion described above, the single CIRCULATING packet traverses all the sensors within the connected component of sensor network. When the CIRCULATING packet visits each node at the first time, the necessary information about this node, including ID field and Incoming/Outgoing filed, are recorded into the packet. Finally, the CIRCULATING packet would return back to the root as long as no further nodes failed when the algorithm is applied. Based on the collected information, the root could calculate the tree structure of the sensor network. Moreover, we should also mention, the necessary table at each sensor (such as Parent field, Children List field) to maintain tree structure in the sensor network is also created and updated when the CIRCULATING packet traverses the network. The detailed pseudo-code is provided in Appendix B.

4.3 A Simple Analysis

In this subsection, we conduct a simple theoretical study for the two proposed global algorithms. It would like to help us to estimate their performance approximately and gain an intuitive insight into their pros and cons. We might assume that the constructed tree is a balanced x-tree [7]. Based on this assumption, we are able to borrow the theoretical analysis from the standard graph theory [8] and roughly calculate the performance for the two global algorithms easily:

- For the global flooding algorithm, the communication overhead is O(nlogn), and the latency is O(logn);
- For the global circulating algorithm, the communication overhead is O(n) and the latency is also O(n).

where n is the number of nodes. Those results could be derived easily if the tree is binary search tree, as shown in Ref.[8]. However, even if the tree structure is the a-tree where each node has a children rather than 2 children, the result is still at the same order of complexity as shown above, while the constant factor may be different.

---

1 We acknowledge that this assumption may not reflect the reality. However, it does help to give an intuitive estimation of algorithm performance in ideal case. Moreover, in the Section IV, we conduct the simulation in non-ideal scenarios that may not follow the assumption.
Founded on this rather intuitive study, it seems that an Energy-Latency trade-off exists for global algorithms of tree reconstruction in sensor networks. This phenomenon could be explained by the different numbers of inquiry processes injected into the sensor networks: In global circulating algorithm, only one inquiry process (i.e., single CIRCULATING packet) traverses the whole network in a DFS-like fashion along the newly established tree. Because it visits all the nodes in one-by-one way, the communication will not be wasted on redundant transmissions but the latency is expected to be high. In opposition, in global flooding algorithm, multiple inquiry entities (i.e., multiple FLOOD packets / SOLICIT packets) are querying all the nodes simultaneously. Hence, the latency is reduced while the communication overhead is unavoidably spent on redundant packet retransmission.

V The Localized Algorithm

Having conducted the simple comparison between the performances of two global algorithms, we find that the global algorithms might not be the appropriate solution. One of the energy-efficient alternatives for the infrastructure maintenance problem in face of node failure is to use the localized algorithm, which checks the status of small portion of all the sensors and only repair the subtree structure in the region where sensors fail. Hence, the communication overhead and latency could be reduced. Specifically, in the localized flooding algorithm discussed below, we restrict the region of inquiry and flooding to the subtree rooted at the K-level ancestor of the failed nodes.

5.1 Observations and Basic Idea

The proposition of the localized algorithm is based on the two observations we made in the simulations:

1. The node density of sensor network is high. If so, even one node fails, its children could still choose other neighbors alive as the new parents and reconnect to the damaged tree through those new parents.
2. The ‘family’-closeness in the logical tree structure for sensor network may also implicitly indicate the geographic-nearness between sensors. If sensors locate within each other’s transmission range, it is highly possible that some sensors are the parents, or children, or siblings, or cousins of the other sensors.

Thus, after some sensors fail, for the tree structure, all the nodes within the subtree rooted at this failed sensor may become ‘soundless’, since their previous routes to root are broken. However, because of the properties of the high-density deployment and implicit logical-geographic relationship mentioned above, those soundless nodes could possibly find other living neighbors located in other branches of the existing tree structure as the new potential parents, and attach to the tree structure through those nodes. For example, in Figure 2(c), node A fails but all the nodes within the subtree rooted at node A (including B and C) can be attached to the tree structure through node D.

In our localized flooding algorithm, when the node failure is detected, a high-level ancestor of the failed node is notified about the failure. Then, this notified ancestor would conduct a localized flooding for the nodes in its subtree (of the previously established tree structure). Thus, this flooding is only restricted to a small portion of sensor field and the whole tree is not needed to rebuild from scratch. In other words, we only repair the tree structure at the region where sensors fail. Hence, the overhead and latency is not wasted to inquiry about the node status and reconstruct the tree structure for intact part of sensor network.

5.2 Localized Flooding Algorithm

For localized flooding algorithm, the packet types used and the table entries kept at sensor are similar to the global flooding algorithm. The only difference is that, for the localized flooding algorithm, we add the new Isolation field and Restriction field in the table, as shown in Figure 1(c). For the Isolation field, white color represents that the node is currently not part of established tree structure of sensor network, i.e., this sensor is soundless or so-called ‘isolated’ node. Black color indicates that the node is part of already established tree that are not affected by the node failure; For the node with white color for Isolation field, its Restriction field is invalid. Only for the node whose Isolation field is black, the value of Restriction field is meaningful: the NO value represents that the node should not be used for localized flooding, the default value of Restriction field is set as NO. The YES value indicates that the node is part of already established tree structure (black color for Isolation field) but through them the isolated nodes may potentially reconnect to the tree structure. In other words, the nodes with YES value for Restriction field are the only nodes used for localized flooding.

When the failure of sensor is detected, the downstream detectors of the failed sensors will send ERROR packets to all the children in their subtrees, in order to notify them to change their Isolation field to white and invalidate their Restriction/Parent/Children/Level field. Soon, all the nodes within the subtree rooted at failed sensor are labeled as ‘isolated’ nodes whose Isolation field is white. At the same time, the failure of sensor is also detected by the upstream
detector. The upstream detector will report the failure in an ERROR packet to its K-level ancestor (rather than root), where the fixed parameter ‘K’ is the key parameter in our localized flooding algorithm. It represents to which level of ancestor should be notified about the lower-level sensor failure and from which level of ancestor the localized flooding should be conducted, we call the K-level ancestor of the failed sensor as flood initiator because the localized flooding is initiated by it. If K is 1, then the flood initiator is the upstream detector itself; if K is 2, then the flood initiator is the parent of upstream detector; if K is 3, the flood initiator is the grandparent of upstream detector, so on and so forth.

After receiving the ERROR packets from upstream detector, the flood initiator will initiate the localized flooding by broadcasting FLOOD packets within its subtree. For the black node within this subtree, after receiving the FLOOD packets from its parent, it changes its Restriction field to YES, while the nodes in other subtree remain NO value. By this way, for all the black nodes within the subtree of flood initiator, their Restriction field will be changed to YES. Finally, the FLOOD packets may reach ‘isolated’ nodes whose Isolation field is white in the silent zone. When the white node receives the FLOOD packet, it will adopt the node from which the FLOOD packet comes first as the parent, and then reply a SOLICIT packet towards the newly adopted parent. Thus, the ‘isolated’ node could be reconnected to tree structure. After changing the Isolation field to black and label its Restriction field as YES, this newly reconnected node will rebroadcast the FLOOD packet in its transmission range. This procedure is repeated until no white nodes exist.

Once the nodes outside of this subtree rooted at flood initiator receive the FLOOD packet, the FLOOD packet is dropped because this packet is not forwarded from their own parents in the tree, so the value of Restriction field is still set as default value NO. In this fashion, the localized flooding is restricted within the subtree rooted at the flood initiator, and intact part of damaged tree can be reused without incurring much overhead. For this reason, the tree structure is said to be repaired rather than to be re-established from scratch, as in global algorithm. The example is shown in Figure 2(c). For the detailed pseudo code, please refer Appendix C.

Apparently, the selection of K parameter would impact the performance of our localized flooding scheme: Within the previously established tree structure which is damaged now, the K parameter decides the position from which the localized flooding is initiated, which in turn indirectly determines the region where the restricted flooding is conducted. Suppose that the failed sensor is fixed, if the K parameter is smaller, the position of flood initiator is at lower level of the tree structure. As a result, the nodes covered by localized flooding are expected to be small, so is the overall communication overhead and latency. On the contrary, if the K parameter is a large number, the position of flood initiator would be close to the root of the tree, so the region covered by localized algorithm is larger. Consequently, the communication overhead and latency will increase as the K parameter increases. In summary, from the perspective of energy, small value of K parameter is desirable.

However, on the other hand, small K parameter may result in the incorrect or non-optimal tree restoration. In some scenarios, if the K parameter is too small, the ‘isolated’ nodes cannot be reconnected to the existing structure through the small-range localized flooding; because the only nodes that could reconnect the isolated nodes may not locate in the region of localized flooding; Similarly, the nodes from which the ‘isolated’ nodes reconnect to the tree structure in shortest-path-length fashion also might not locate in the region of localized flooding as well. Thus, even the isolated nodes could be reconnected to the tree structure, their path lengths to base station will not be optimal. If the K parameter increases, the instances of incorrect and non-optimal tree restoration are reduced because the region of localized flooding is expanded.

Based on the above intuitive discussion, we recognize the relationship of K parameter and the performance of localized flooding algorithm in terms of various metrics. In the next section, we analyze this relationship in details by simulation.

VI The Simulations and Results
In this section, we report the preliminary evaluation of various scenarios. In addition, we also gain a deeper understanding towards the Energy-Latency trade-off for global algorithms and the Energy-Accuracy/Efficiency trade-off for localized algorithm.

6.1 Simulation Setting and Metrics
The result of this section is obtained in an event-driven packet-level simulator developed by the authors. We implement the proposed two global algorithms and one localized algorithm in this simulator. This simulator uses the ideal radio transmission model with transmission range $R = 150m$ and the ideal MAC layer model. In order to study the performance trend of proposed algorithms as the function of network size, the number of sensor nodes is set to 50, 100, 150, 200 and 250 to generate a number of sensor networks. The nodes are randomly and uniformly placed in the field. As noticed by many other previous works[2,3], the node density is expected to be high in sensor network. We assume that the generated sensor networks are keeping an approximate average node degree at around 10. Thus, the sensor network with 50 nodes is generated in a 600m by 600m square field, and 1275m by 1275m field can contain a 250-node sensor network. The failure of sensor is modeled as sporadic failure, i.e., the failure of sensor is random and independent of each other. In our simulator, at every second, one sensor among all the nodes alive is randomly and uniformly chosen to fail with probability $p$ ($p=0.4$ in our simulation setting). The simulations keep running until 30% of the overall nodes in the sensor field fail.

---

2 If the K-level ancestor of failed node does not exist (i.e., the K-m level ancestor of failed node is already the root, where m is smaller than K), we will choose the root as the flood initiator.
For each network size, we generated 5 different instances of network topology with the different random seeds. For each generated network topology, we generate 6 different kinds of node failure scenarios with various random seeds. In short, for each network size, the final result shown here is averaged over 30 different instances.

We choose two major metrics to evaluate the performance of three proposed algorithms: Average Communication Overhead and Average Latency to reconstruct the tree structure. The Communication Overhead metric measures the number of overhead used to inquiry about the node status and reconstruct the tree structure, including ERROR, FLOOD, SOLICIT, CIRCULATING packets etc. It is counted as the number of transmitted and forwarded packets. The Latency metric measures the latency observed as the difference between when node failure is detected and when the tree structure is restored. It is counted in terms of timestep.

To study the relationship between K parameter and performance of localized algorithm, besides the metrics of overhead and latency, we also examine the metrics of incorrectness and non-optimality of reconstructed tree by localized flooding algorithm. As pointed in section 5.2, if K parameter is too small, localized flooding algorithm may not always correctly reconstruct tree structure to route packets optimally: Some soundless nodes may hardly be reconnected to the tree structure. Since the localized algorithm could not decide whether it is error case. By comparing with the correctly repaired tree structure obtained from global flooding algorithm, we are able to tell whether it is the error case. The metric of Incorrectness is defined as the ratio of error case happens; In other cases, some soundless nodes may fail to reconnect to the tree through the shortest-path route. Thus the metric of Non-optimality is defined as the ratio of non-optimal-path case happens.

6.2 Performance Comparison for Proposed Algorithms

In this subsection, we look at the general performance trend for various algorithms. For the localized flooding algorithm, we set K parameter equal to 1 as the baseline performance of this algorithm (i.e., the upstream detector itself will initiate the localized flooding). In the next subsection, by varying the K parameter, we will examine the detailed relationship between localized algorithm performance and K parameter.

Figure 3(a) shows the trend of Average Communication Overhead incurred by each node failure for these algorithms. Clearly seen from the figure, localized flooding algorithm achieves the lowest communication overhead. Between the two global algorithms, the global circulating algorithm generates much less communication overhead than the global flooding algorithm. Thus, the global flooding is the most expensive algorithm in terms of communication cost and corresponding energy. The trend of Average Latency for the three algorithms is illustrated in Figure 3(b). The localized algorithm also outperforms the global algorithms in terms of latency. However, between the global algorithms, the latency incurred by the global circulating algorithm is higher in the order of magnitude than the global flooding algorithm.

In summary, the simulations validate our conjecture that localized algorithm outperforms the global algorithms in terms of both overhead and latency. This is because the localized algorithm only inquiries the small portion of nodes around the failed node, while the global algorithm needs to check all the nodes alive.

6.2 Energy-Accuracy/Optimality Trade-off for Localized Flooding Algorithm

Prior to investigating the impact of K parameter on the performance metrics for localized flooding algorithm, we may first examine how the smallest K parameter (i.e., K=1, used as the baseline value) determines the performance in terms of correctness and route optimality. When K is equal to 1, i.e., the localized flooding is only restricted to the subtree rooted at the upstream detector of failed sensors, the localized algorithm may generate around 7%~11% error cases among all the scenarios with different network sizes. The case of non-optimal route happens around 10%~12% for all cases. However, by tracing the non-optimal routes established by localized flooding algorithm, we find that the majority of non-optimal routes
are only 1~2 hop longer than the optimal routes (the maximal path length is 14 hops). When the K is set to 2, the error case decreases to less than 6% and non-optimal case decreases as well.3

Then, we might look at how the K parameter can determine the performance of algorithm in terms of incorrectness and non-optimality together with overhead and latency. As discussed in section 5.2, the value of K parameter indirectly indicates the large region of localized flooding. As the value of K parameter increases, the region of localized flooding is expanded, then the communication overhead and latency will increase as well. However, the incorrectness and non-optimality could be improved at the same time, since it begins to approximate the global flooding. As shown in Figure 4(a), the overhead (and corresponding energy) increases linearly as K increases. At the same time, as shown in Figure 4(b), the error probability and non-optimization probability decreases sharply. As K parameter increases, the latency also increases linearly. Due to the limited page, we did not show how parameter K affects the latency.

By observing those figures, we are able to develop a deeper understanding for the Energy-Accuracy/Optimality trade-off for localized algorithm. When K parameter is set to 1, the localized flooding algorithm only incurs around 1/15 overhead of the global flooding algorithm, however, it will also generate around 11% error cases and less than 12% non-optimal routes. When the parameter is set to 2, the overhead incurred by localized algorithm is around 1/9 of the global flooding algorithm, but the error case and non-optimal case is reduced to 6% and 9% respectively. In sensor network, we may sacrifice the accuracy or optimality of result to save the scarce energy, as observed in previous works [9].

![Figure 4(a) Impact of K on Overhead (N=250)](image)

![Figure 4(b) Impact of K on Incorrectness and Non-optimality (N=250)](image)

VI Conclusion & Future Work

The tree structure used as the underlying infrastructure for sensor network is subject to change when some sensors fail. In this paper, we attempt to provide solutions to re-establish or repair the broken tree structure in an efficient fashion.

Two global algorithms, including global flooding algorithm and global circulating algorithm, as well as one localized flooding algorithm are proposed. Because the whole tree structure is constructed from scratch and all nodes are inquired about status, the global algorithms are rather expensive in terms of energy, compared with localized algorithm. For localized flooding algorithm, we find that the goal of reducing energy and goal of improving correctness (or routing optimality) are somehow contradict, while both of them are determined by the region of restricted flooding (i.e., K parameter in our localized flooding algorithm). Moreover, we observe that the Energy-Latency trade-off (for global algorithm) and the Energy-Accuracy/Optimality trade-off (for localized algorithm) seem to exist in sensor network.

As part of our future work, one immediate goal would be to develop a scheme where K parameter is adaptively adjusted by the network. According to the level of failed sensor in the tree structure, node degree and other parameters, the K-parameter is dynamically chosen to achieve the appropriate Energy-Accuracy/Optimality trade-off observed in this paper.

[Reference]

3 As a side note, we should mention that the incorrect and inefficient tree restoration would not be accumulated over time. Because the sensor failure randomly happens to every sensor, thus some sensor close to root could be expected to fail over some time. At this moment, a global flooding initiated by root (which is just the flood initiator) could restore the tree structure efficiently and correct ly. By tracing the details, the claims are observed in our simulations.
Appendix A The pseudo-code for global flooding algorithm

global_flood ( )
{
    for each node i, when it receives a packet from some other nodes
        if (packet.type == ERROR){
            if (node[i].id != root )  // if the node is not root, forward it towards root
                unicast(packet, node[i].parent);
            else {       // if it is root, increments the seq_num, and initiates new flooding
                root.seq_number ++;
                form_packet( new_packet, FLOOD);
                packet.seq_number = root.seq_number;
                broadcast( new_packet );
            }
        }
        if (packet.type == FLOOD){
            if (packet.seq_number > node[i].seq_number)
            {
                // if the FLOOD packet is new FLOOD packet, update the seq_number, set the parent
                // as the node where the packet comes, and rebroadcast the packet in its neighborhood
                node[i].seq_number = packet.seq_number;
                node[i].parent = incoming_id(packet);
                node[i].level = node[node[i].parent].level+1;
                broadcast(packet);
            }
        }
        if (packet.type == SOLICIT){
            if (incoming_id(packet).parent == node[i].id ){  //if the node is the parent, establish the parent-children relationship
                add_children (node[i], incoming_id(packet));
            }
            if (node[i].id != root )  //forward this packet until it reaches at root
                unicast (packet, node[i].parent);
        }
    }
}

The pseudo code above describes the major part of the algorithm of global flooding. Some functions are used in it. Here, we briefly describe their functionalities: form_packet (packet, type, seq_number) is to form a new packet with type as parameter ‘type’ and the parameter ‘seq_number’ as the sequence number of the packet; add_children (parent, child) is to add the node ‘child’ into the children list of node ‘children’; incoming_id (packet) is to distinguish the node from which the packet is sent; unicast (packet, receiver) is to send the packet to the node ‘receiver’ and broadcast (packet) is to broadcast the packet in the transmission range of the node.
Appendix B  The pseudo-code for global circulating algorithm

```c
Problem: Given a network of nodes, design an algorithm to identify and repair a broken link.

Algorithm:

1. Each node maintains a sequence number and a status (BLACK, WHITE, GRAY).
2. When a node receives a packet, it checks the packet type:
   - If the packet is an ERROR, it forwards it to the root node if it is not the root itself.
   - If the packet is a CIRCULATE, it checks the sequence number and status of the node:
     - If the packet has been seen before, drop it.
     - If the node is BLACK, drop it.
     - If the node is WHITE, update the sequence number and parent/level fields.
     - If some neighbors are still WHITE, send the packet to one of them.
   - If the node is GRAY, add the incoming node as its child node.
     - If some neighbors are still WHITE, send the packet to one of them.
     - If none of the neighbors is WHITE, change its status to BLACK and return.
3. The root node initiates the circulating process when it receives an ERROR packet.
```

In the pseudo-code above, we describe the major part of the global circulating algorithm. Some of the functions are described as follow: drop_packet(packet) is to drop the packet because it is mistakenly forwarded; neighbor_list(i) function
```
returns the pointer to the array which maintains all the neighbors of node i whose status is still white. If none of such neighbor exists, it returns NULL value. The other functions are similar to the functions described in Appendix A.

Appendix C  The pseudo-code for localized flooding algorithm

```plaintext
error_handle ( K-parameter )
{
    for the downstream detector
    while (subtree_of_downstream_detector (i)) { //to invalidate all nodes in the silent zone
        unicast(node[i].children_list, error_packet);
        node[i].isolation = WHITE;
        node[i].restriction = NULL;
        node[i].parent = NULL;
        node[i].children_list = NULL;
        node[i].level = NULL;
    }

    for the upstream detector
    while ( ! K-ancestor_of_upstream_detector (i) ) { // notify the ERROR to flood initiator
        unicast (node[i].parent, error_packet);
        i = node[i].parent;
    }

    flood_initiator = i;
    //then flood initiator initiates the localized flooding
    packet.type = FLOOD;
    unicast (node[flood_initiator].children_list, packet);
}

local_flood ()
{
    for each node i in the network, when it receives a packet from some other nodes
    if (packet.type == FLOOD){
        if( subtree_of_flood_initiator(i) )
            if( node[i].isolation == BLACK){
                //for black nodes, only enable their restriction field, and broadcast
                node[i].restriction = YES;
                broadcast(packet);
            } else if( node[i].restriction == WHITE) {
                //for white node, record the field used to establish tree structure
                //enable the restriction field, change isolation field, and rebroadcast FLOOD
                //send the SOLICIT packet to parents.
                node[i].isolation = BLACK;
                node[i].restriction = YES;
                node[i].parent = incoming_id(packet);
                node[i].level = node[node[i].parent].level + 1;
                broadcast(packet);
                form_packet(new_packet, SOLICIT);
                unicast(new_packet, node[i].parent);
            }
        } else ( !subtree_of_flood_initiator(i) )
            drop_packet(packet);
```
if (packet.type == SOLICIT) {
    if (incoming_id(packet).parent == node[i].id) {
        // if the node is the parent, establish the parent-children relationship
        add_children (node[i], incoming_id(packet));
    }
    if (node[i].id != root) {
        // forward this packet until it reaches at root
        unicast (packet, node[i].parent);
    }  
}

In the pseudo-code above, we describe the major part of the global circulating algorithm. Some of the functions are described as follows: subtree_of_downstream_detector(i) is to decide whether node i is within the subtree of the downstream detector, if so, return TRUE, otherwise, it returns FALSE; similarly, subtree_of_flood_initiator(i) is to decide whether node i is within the subtree of the flood initiator. K-ancestor_of_upstream_detector(i) is to decide whether the node i is the k-level ancestor (i.e., flood initiator) of the failed node in the tree structure.