Data Dissemination with Ring-Based Index for Wireless Sensor Networks

Wensheng Zhang
Department of Computer Science
Iowa state University
Ames, IA 50010
E-mail: wzhang@cs.iastat.edu

Guohong Cao and Tom La Porta
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
E-mail: {gcao, tlp}@cse.psu.edu

Abstract

In wireless sensor networks, sensor nodes are capable of not only measuring real world phenomena, but also storing, processing and transferring these measurements. Many techniques have been proposed for disseminating sensing data. However, most of them are not efficient in the scenarios where a huge amount of sensing data are generated, but only a small portion of them is queried. In this paper, we first propose an index-based data dissemination scheme to address the problem. With this scheme, sensing data are collected, processed and stored at the nodes close to the detecting nodes, and the location information of these storing nodes is pushed to some index nodes, which act as the rendezvous points for sinks and sources. To address the issues of fault tolerance and load balance, we extend the scheme with an adaptive ring-based index (ARI) technique, in which the index nodes for one event type form a ring surrounding the location which is determined by the event type, and the ring can be dynamically reconfigured. Considering that frequently updating or querying index nodes may cause high overhead, we also propose a lazy index updating (LIU) mechanism and a lazy index querying (LIQ) mechanism to reduce the overhead. Analysis and simulations are conducted to evaluate the performance of the proposed scheme. The results show that the proposed scheme outperforms the external storage-based scheme, the DCS scheme, and the local storage-based schemes with flood-response style. The results also show that using ARI can tolerate clustering failures and achieve load balance, and using LIU (LIQ) can further improve the system performance.

1 Introduction

Recent advances in electronic and communication technology have enabled the production of battery-powered wireless sensor nodes. These nodes are capable of not only measuring real world phenomena, but also storing, processing and transferring these measurements. They are typically deployed in the physical world to form a wireless sensor network [3], which enables many applications such as battlefield surveillance, habitat monitoring [7, 26], disaster rescue, and traffic tracking. These applications involve collecting, processing, storing and retrieving a large volume of sensing data generated by the sensor nodes. Since bandwidth and power are scarce resources in wireless sensor networks, it is critical to design scalable and energy efficient data dissemination schemes.

In the past, many data dissemination schemes [13, 14, 29, 20, 10, 11] have been proposed for sensor networks. One widely adopted approach is the *external storage-based (ES)* scheme. It relies on a centralized server, which is outside of the sensor network, for collecting and storing sensing data. As an alternative approach, the *data-centric storage-based (DCS)* scheme was proposed by Ratnasamy *et al.* [20]. In this scheme, events to be detected are named, and the sensing data of these events are stored at nodes within the network, instead of being forwarded to an external storage. As shown in Figure 1 (a), the storing node of an event is the node closest to a location calculated by applying a hash function on the event type. No matter sensing data are pushed to the storage outside of the network or that within the network, in the above approaches, all the data must be transferred regardless of whether they are used by the applications. Hence, they lack flexibility and may introduce lots of unnecessary data transfers.

Unnecessary transfers of sensing data can be avoided in the *local storage-based (LS)* data dissemination schemes, e.g., the directed diffusion [14] and the two-tier data dissemination (TTDD) [29], in which a source sends data to a sink only when the sink has queried the data. These schemes, however, need a sink-source matching mechanism to facilitate a sink in finding the source that holds the data of interest. The matching mechanisms adopted by most LS schemes follow a *flood-response* pattern [11], which inherently needs to flood certain control messages. For example, in the directed diffusion, a sink may have to flood its query over the whole network; the sources with the requested data then

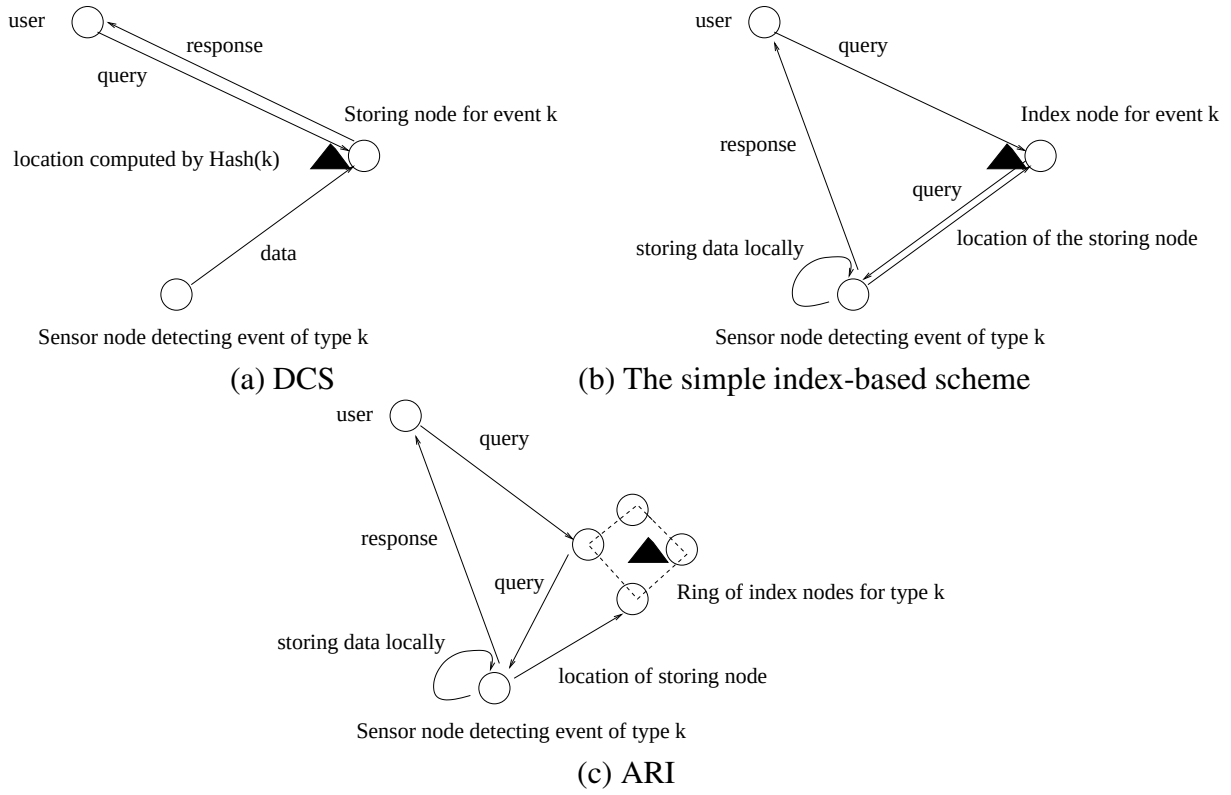


Figure 1: Data dissemination schemes

knows where to send the data. In TTDD, the source detecting a certain event floods the advertisements of the event to the network, and the sinks interested in the event can send their queries directly to the source. Considering the large number of nodes in a sensor network, the network-wide flooding may incur significant traffic.

Due to the aforementioned reasons, the existing ES, DCS and LS data dissemination scheme may not work well in large scale sensor networks, especially in the scenarios where a large amount of sensing data are generated, but only a small portion of them will be queried. To address this problem, we propose an index-based data dissemination scheme. In this scheme, as shown in Figure 1 (b), the sensing data of an event are stored at the detecting nodes themselves or some nodes close to them (called *storing nodes*). A storing node sends data to a sink only when it receives a query from the sink. Also, the location information (called *index*) of the storing nodes is pushed to and maintained at some nodes (called *index nodes*) based on the event type of the stored data. Hence, queries for a particular event are directly routed to the appropriate index nodes, which may then forward the queries

to appropriate storing nodes. This scheme avoids both unnecessarily transferring the sensing data and flooding control messages throughout the network. Certainly, this scheme introduces additional overhead for maintaining index nodes. However, as demonstrated by the analysis and evaluation results in this paper, it can still improve the overall system performance.

One big challenge in implementing the index-based data dissemination is how to maintain the indexes in the network, such that the indexes are highly accessible to sinks and the index nodes are not overloaded. To achieve these goals, we propose an approach based on the *ring* structure, and call it the *Adaptive Ring-based Index (ARI)* scheme [30]. In ARI, as shown in Figure 1 (c), multiple nodes are picked as the index nodes for each event type, and these index nodes connected with each other to form a ring (called *index ring*). The number and locations of the index nodes on an index ring, as well as the shape of the ring, can be adaptively changed to tolerate node failures and achieve load balance.

Based on the ARI scheme proposed in [30], we further propose in this paper two enhancements to reduce the structure maintenance overhead. These enhancements are based on the following observations: With ARI, a source needs to update its location at the index nodes whenever its location changes. If a target frequently moves, its source node also frequently changes, and the overhead of index updating may become very high. Furthermore, most of the updates may be useless if the source is seldom queried. On the other hand, some sinks may continuously monitoring a moving target by frequently querying the source associated with the target. Using ARI, the sink needs to query the index nodes before querying the storing node. Therefore, the overhead of query can also be high. To deal with the above drawbacks, a *lazy index updating (LIU)* mechanism and a *lazy index query (LIQ)* mechanism are designed to reduce the overhead, and thus improve the overall system performance.

Extensive analysis and simulations are conducted to analyze and evaluate the performance of the proposed index-based data dissemination scheme, and its performance is compared to the existing data dissemination schemes. The results show that, the index-based scheme outperforms external storage-based schemes, the DCS scheme, and local storage-based schemes in many scenarios. The results also show that using the ARI scheme can tolerate clustering failures and achieve load balance, and the proposed enhancements can further improve the system performance.

The rest of the paper is organized as follows. Section 2 presents the preliminaries. Section 3 describes the index-based data dissemination scheme and analyzes the overhead of several data dissemination schemes. Section 4 describes the proposed ARI scheme in detail, and Section 5 presents some enhancements to ARI. Performance evaluations are presented in Section 6. Section 7 presents some related work. Finally, Section 8 finally concludes the paper and proposes some new directions for future research.

2 Preliminaries

In this section, we first describe the system model that we are considering. Then, we present the expected application scenarios for our proposed scheme.

2.1 System Model

We consider a large-scale sensor network that is deployed to a vast field. All the sensor nodes are stationary, and are aware of their own locations using GPS [2] or other techniques such as triangulation [6]. To save power, the nodes stay in the sleep mode during most of the time based on the Geographical Adaptive Fidelity (GAF) protocol [28]. Using this protocol, as shown in Figure 2, the field covered by the network is divided into small virtual grids. The virtual grid is defined such that, for any two adjacent grids, all nodes in one grid can communicate with all nodes in the other grid and vice versa. In each grid, nodes rotate to keep awake and act as *grid head*, while others only need to wake up periodically. The grid head is responsible for forwarding messages that pass through the grid.

When data or control messages are forwarded in the network, the Greedy Perimeter Stateless Routing (GPSR) [4, 15] is employed. GPSR uses two distinct algorithms for routing. One is a greedy forwarding algorithm which forwards packets progressively closer to the destination at each hop. The other is a perimeter forwarding algorithm that forwards packets where greedy forwarding is impossible. In our system, which uses both the GAF protocol and the GPSR protocol, packets are forwarded by grid heads. A grid head first tries to forward a packet to its neighboring grid head

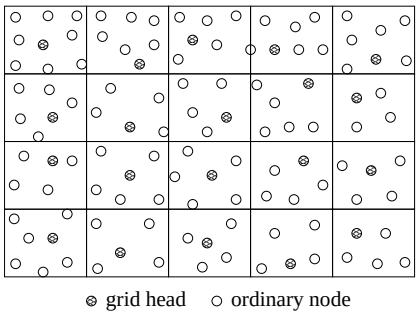


Figure 2: Dividing a sensor network into grids

that is closest to the destination. On failing to find such a node, it forwards the packet to one of the neighboring grid heads based on the perimeter forwarding algorithm.

2.2 Application Scenarios

Our design aims to support efficient data dissemination for the application scenarios where a large amount of sensor data are generated but only a small fraction of them are queried by users. For instance, a sensor network may be deployed to an open area for security monitoring. The network needs to detect and record the activities of all types of targets (human beings, animals, vehicles, *etc.*) that have entered the monitored area. Therefore, a large amount of sensing data will be generated. However, most of the data will not be used. The typical usage of the data can be as follows: for the purpose of precaution, the current status of each type of target is queried every certain time interval; only when an intrusion has been detected, detailed information of the related targets (e.g., the targets that are of certain target types and are located in a certain area) is retrieved for full investigation. Similar examples can also be found in the applications such as animal habitat monitoring, battlefield surveillance, and so on. In such scenarios, typical queries can be expected to be spatial-temporal and/or related to certain target types. For example, *how many targets of a particular type are within a particular geographic area during a particular time period* and *what is the current status of the target that is of a particular type and was in a particular area during a particular time period*.

When designing our data dissemination scheme, we assume that sensor nodes can classify the targets that they detect to one of predefined types, and can track these targets. These problems of

target classification and tracking have been studied in an ample set of prior work [5, 12, 24, 32, 27, 19]. Once the targets are classified, they can be given unique names. Let us consider a specific scenario, where the sensor network is deployed to some wild field for monitoring the habitat of animals. We can predict that the targets that will enter the field may include some known animals, human beings, and vehicles. Assuming the collaborative signal processing framework proposed in [5] is used for target classification, the network should first be trained to classify the above targets into types (classes), and each target type is associated with a unique name (which could be an integer). For example, the name for bears is 1, the name for elephants is 2, and so on. Sensor nodes keep the mapping between the target types (i.e., signal patterns obtained from the training) and names. Note that the network administrator also maintains the mapping between target types and names, so that users interested in a certain target can obtain the corresponding name from the administrator, and then send to the network their queries based on the name. Given the size of the monitoring field, a hash function $H(.)$ can be constructed which maps each target type to one location within the field. For scalability, if the number of names is large, we can merge multiple types into a type group (which could also be named as an integer), and the hash function will map each type group to one location. Formally, $(\forall e \in E)H(e) = L_e \in \mathcal{R}$, where E is the set of names (corresponding to types or type groups) and $\mathcal{R}$ is the monitoring region.

3 The Index-Based Data Dissemination

In this section, we first describe the basic idea of the index-based data dissemination. Then, we analytically compare the overhead of different data dissemination schemes, and identify the scenarios in which the index-based scheme performs the best.

3.1 Basic Idea of the Index-based Data Dissemination

The basic idea of the index-based data dissemination is illustrated in Figure 3 and is described as follows: A node detecting a target periodically generates sensing data about the target, and stores the data in a storing node, which can be itself or some node nearby. When the target moves, as shown in

Figure 3 (b) and (c), the detecting node is changed accordingly. However, the new detecting node still stores the sensing data at the same storing node, until the storing node is far away from the detecting node, and then a new storing node is selected. When a new storing node of a target has been selected, the new storing node will keep a pointer which refers to the old one, such that the previous data can also be accessed from the current storing node. Also, the new storing node should register its location at the index nodes for the target. When a sink wants to query the detailed sensing data of a target belonging to a certain type, it sends a query message to an index node for the type. On receiving the message, the index node forwards the request to the corresponding storing node which sends a response directly to the querying sink.

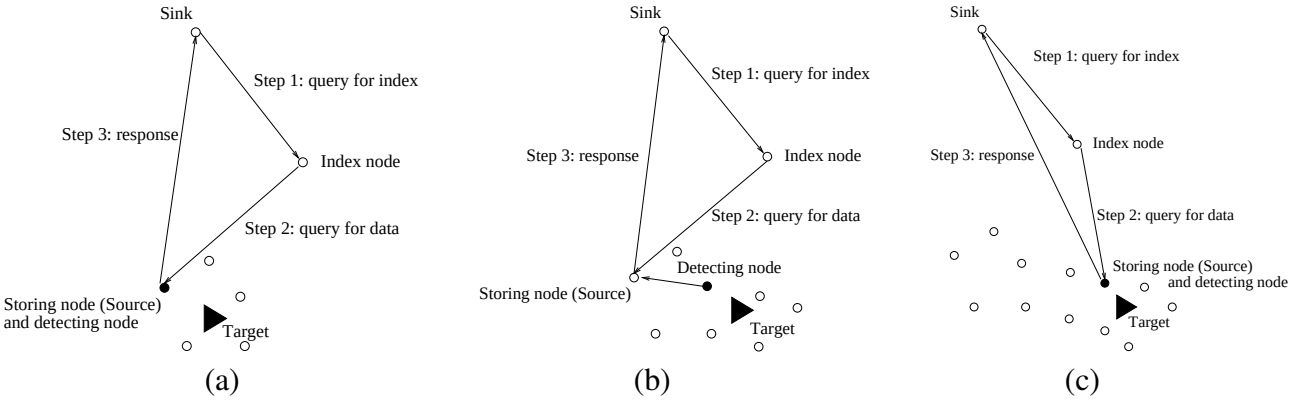


Figure 3: Index-based data dissemination

3.2 Analytical Comparison of the Data Dissemination Methods

To compare the overhead of the index-based scheme with some other data dissemination schemes, we analyze the cost for processing queries. Before presenting the analysis, we introduce some notations as follows:

- N: the total number of sensor nodes in the network. Similar to [20, 10], we use $O(N)$ to represent the message complexity of flooding a message to the whole network, and use $O(\sqrt{N})$ to represent the message complexity of sending a message between two nodes in the network.

- T, n: T is the number of types, and n is the number of targets of each type.

Table 1: Estimating the overhead of the data dissemination schemes

Scheme	Overall message complexity	Hotspot message complexity
ES	(1.1) $O((\sqrt{N} * r_d + 2 * \sqrt{N} * r_q) * T * n)$	(1.2) $O((r_d + 2 * r_q) * T * n)$
DCS	(2.1) $O((\sqrt{N} * r_d + 2 * \sqrt{N} * r_q) * T * n)$	(2.2) $O((r_d + 2 * r_q) * n)$
LS	(3.1) $O((N * r_c + 2 * \sqrt{N} * r_q) * T * n)$	(3.2) $O(r_c * T * n + 2 * r_q)$
Index	(4.1) $O((\sqrt{N} * r_s + 3 * \sqrt{N} * r_q) * T * n)$	(4.2) $O((r_s + 2 * r_q) * n)$
Scheme	Overall traffic	Hotspot traffic
ES	(1.3) $O(\{\sqrt{N} * r_d * s_d + \sqrt{N} * r_q * (s_q + s_d)\} * T * n)$	(1.4) $O(\{r_d * s_d + r_q * (s_q + s_d)\} * T * n)$
DCS	(2.3) $O(\{\sqrt{N} * r_d * s_d + \sqrt{N} * r_q * (s_q + s_d)\} * T * n)$	(2.4) $O(\{r_d * s_d + r_q * (s_q + s_d)\} * n)$
LS	(3.3) $O(\{N * r_c * s_i + \sqrt{N} * r_q * (s_q + s_d)\} * T * n)$	(3.4) $O(r_c * s_i * T * n + r_q * (s_q + s_d))$
Index	(4.3) $O(\{\sqrt{N} * r_s * s_i + \sqrt{N} * r_q * (2 * s_q + s_d)\} * T * n)$	(4.4) $O(\{r_s * s_i + r_q * (s_q + s_d)\} * n)$

- r_d, r_q, r_c, r_s: r_d is the rate of updating the sensing data of a target. r_q is the rate of querying a target. r_c is the rate that the detecting node of a target is changed. r_s is the rate that the storing node of a target is changed, where $r_d \geq r_c \geq r_s$. In most cases, $r_d > r_c > r_s$.

- s_d, s_q, s_i: s_d is the size of a data report, s_q is the size of a query message, and s_i is the size of an index update message. In most cases, $s_d \gg s_q$ and $s_d \gg s_i$.

We consider the following metrics when comparing the overhead of the data dissemination schemes:

- **Overall message complexity**: the total number of hops that all the messages must be forwarded. For example, if m messages are transferred and each of them must go through h hops, then the overall message complexity will be $m * h$.

- **Hotspot message complexity**: the maximum number of messages sent, received and forwarded by one single node in the network.

- **Overall traffic**: the amount of data transferred in the whole network.

- **Hotspot traffic**: the maximum amount of data sent, received, and forwarded by one single node in the network.

According to the basic operations of the index-based scheme and the other data dissemination schemes [20], we can estimate the overhead of the data dissemination schemes in terms of the overall

message complexity, hotspot message complexity, overall traffic and hotspot traffic. The estimated results are shown in Table 1.

Based on the estimations, we can obtain the following observations:

Observation 1: The index-based scheme has smaller overall message complexity than the other schemes, if:

(1) N is large enough.

(2) $r_s + r_q < r_d$, which can be derived by comparing (4.1) to (1.1) and (2.1).

Observation 2: The index-based scheme has smaller hotspot message complexity than the other schemes, if

(1) $\frac{r_q}{r_c} < T/2$. This relationship is derived as follows:

Let $(r_s + 2 * r_q) * n < r_c * T * n + 2 * r_q$ and $r_s = c * r_c$, where $c < 1$.

Thus, $\frac{r_q}{r_c} < \frac{n*(T-c)}{2*(n-1)} \approx T/2$.

(2) $r_s < r_d$, which can also be derived by comparing (4.2) to (1.2) and (2.2).

Observation 3: The index-based scheme has smaller overall traffic than the other schemes, if:

(1) N is large enough.

(2) $r_s * s_i + r_q * s_q < r_d * s_d$, which can be derived by comparing (4.3) to (1.3) and (2.3).

Observation 4: The index-based scheme has smaller hotspot traffic than the other schemes, if:

(1) $\frac{r_q}{r_c} < \frac{s_i * T}{s_q + s_d}$. This relationship is derived as follows:

Let $\{r_s * s_i + r_q * (s_q + s_d)\} * n < r_c * s_i * T * n + r_q * (s_q + s_d)$ and $r_s = c * r_c$, where $c < 1$.

Thus, $\frac{r_q}{r_c} < \frac{s_i * n * (T-c)}{(s_q + s_d) * (n-1)} \approx \frac{s_i * T}{s_q + s_d}$.

(2) $r_s * s_i < r_d * s_d$, which can be derived by comparing (4.4) to (1.4) and (2.4).

4 An Adaptive Ring-based Index (ARI) Scheme

We now propose an adaptive ring-based index (ARI) scheme. We first present the motivations of the scheme, and then describe the operations including index querying/updating, failure management, and adaptive ring reconfiguration in detail.

4.1 Motivation

The index-based data dissemination is similar to the problem of locating contents in the peer-to-peer systems such as Chord [25], Pastry [23], Tapestry [31], CAN [21], *etc.* In theses systems, each node is assigned a numerical or a d-dimensional identifier, and it is responsible for owning pointers to objects (e.g., files), whose identifiers map to the node's identifier or region. The set of object identifiers assigned to a node is changed as some nodes join or leave the system. Compared with these systems, sensor networks have their unique characteristics. For example, sensor nodes do not have unique identifiers, but can be aware of their own locations. Also, nodes are prone to failures and are constrained in energy supply. Consequently, one challenge of implementing the index-based data dissemination is how to distribute, update, maintain and query the indexes in the network, such that the following requirements are satisfied:

- Fault tolerance: Since sensor nodes are prone to failures [3], we should not rely on a single node to maintain an index. Techniques such as replication should be employed to achieve a certain level of fault tolerance.

- Load balance: Some index nodes may become overloaded. The scheme should remove the overloaded index nodes and add some lightly-loaded index nodes.

- Efficiency: The scheme should not introduce too much overhead since bandwidth and energy are scarce resources in a sensor network.

One simple scheme can be derived directly from the basic DCS scheme. In this scheme, the index nodes for a target are the nodes which form the smallest perimeter surrounding the location

calculated. However, the index nodes cannot be changed except when they are failed, and all queries and index updates for the target are processed by these nodes. Hence, the index nodes may become overloaded soon, especially when the query rate or the index update rate for the target is very high. Also, since the index nodes for an event type are close to each other, the scheme cannot tolerate clustering failures, which may destroy all the index nodes for an event type. A hierarchical scheme similar to the structured replication DCS (SR-DCS) [20] or the resilient DCS (R-DCS) [10] can be used to tolerate clustering failures. In these schemes, the whole detecting region is divided into several subregions, and there is one index node in each subregion. Using these schemes, the index nodes in some subregion can still be overloaded if the number of queries issued from that subregion is very high. In addition, a query may be routed through several levels of index nodes before reaching the valid index node. Tree-based replication of indexes [22], which is already applied in peer-to-peer networks, may also be used in sensor networks. However, this scheme may introduce significant maintenance overhead. For example, the failure of the root or some intermediate nodes in the tree may cause the tree to be reconfigured. Furthermore, the nodes should be aware of the addition, removal and migration of index nodes. Thus, it is a challenge to design index-based schemes to satisfy the fault tolerance, load balance and efficiency requirements.

4.2 The ARI Scheme

To achieve the goals of fault tolerance, load balance and efficiency, we propose the *Adaptive Ring-based Index (ARI)* scheme. The ARI scheme is based on the structure of ring, which includes the index nodes for the targets of the same type, and the forwarding nodes that connect the index nodes. The basic modules in the ARI scheme are ring initialization, index querying, index updating, failure management and ring reconfiguration, which will be explained in this section.

4.2.1 Initializing an Index Ring

We first describe how to initialize an index ring for the targets of a certain type. As mentioned in Section 2.1, the index center for the target is calculated using a hash function, which maps a target to a location within the detecting region. Initially, the index nodes for the targets are m selected nodes

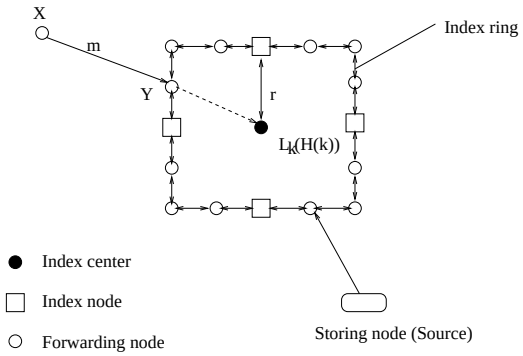


Figure 4: Initializing an Index Ring

whose distance to the index center is equal to or larger than r, where m and r are system parameters. These nodes are connected via some forwarding nodes to form an index ring surrounding the index center. Each node has two pointers which point to its clockwise neighbor and its counterclockwise neighbor on the ring respectively. The index ring must satisfy the following **conditions**:

(*C1*) The distance between the index center and any node on the ring should be larger than or equal to r.

(*C2*) Any message sent by a node outside of the ring-encircled region and destined to the index center, will be intercepted by some node on the ring.

Figure 4 shows an example of an initial index ring, where $m = 4$. The ring satisfies (*C1*), since the distance between each node on the ring and the index center is at least r. Due to the assumption that each node can only forward messages to the nodes in its neighboring grids (refer to Section 2.1), a message sent by a node outside of the ring-encircled region and destined to the index center, must pass some nodes on the ring. For example, as shown in Figure 4, message m sent by node X can be intercepted by node Y. Thus, the ring also satisfies (*C2*). Note that, the index ring may be changed later, due to failures or load balance, but the resulted ring **must also** satisfy the above two conditions.

Parameters m and r affect the system performance. As m becomes large, the number of initial index nodes increases. Hence, the overhead for querying is decreased, at the cost that the overhead for storing and updating index is increased. When r becomes large, the overhead for queries issued by nodes far away from the index center is reduced. Also, more nodes are included in the ring, which

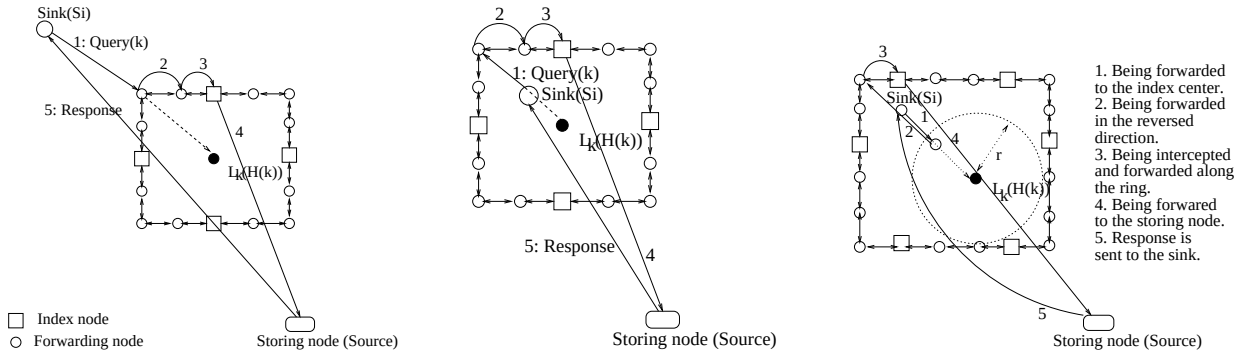


Figure 5: Querying an index

improves the fault tolerance level. However, the overhead for queries issued by nodes within the region encircled by the ring is increased, as is the overhead for index update.

4.2.2 Querying and Updating an Index

The sink S_i can query the index of a target as follows: it first calculates the distance to the index center (denoted as L_k) of the target. If the distance is larger than r, the query message is forwarded along the direction of $S_i \rightarrow L_k$. Otherwise, the message is forwarded along the direction of $L_k \rightarrow S_i$. When a node receives a message, it performs differently based on the following cases:

(1) If the node is an index node for the queried target: the query is forwarded to the storing node of the target if there exists such a storing node.

(2) If the node is a forwarding node on the index ring for the queried target: the query is forwarded to its clockwise neighboring node on the ring.

(3) If the forwarding direction of the message is $S_i \rightarrow L_k$, and the distance between the node and the index center of the target is smaller than r: the forwarding direction of the query message is changed to be $L_k \rightarrow S_i$, and the message is forwarded in the new direction.

(4) Otherwise: the message is forwarded in the specified direction.

Next, we use Figure 5 to further illustrate the querying process. Figure 5 (a) shows the case in which sink S_i is outside of the region encircled by the index ring of the queried target. In this case,

the query issued by the sink is forwarded along the direction of $S_i \rightarrow L_k$, until it is intercepted by some node on the index ring. The node intercepting the query passes the message on the index ring in the clockwise direction, and the query is finally received and served by an index node, which further forwards the message to the storing node of the queried target.

Figure 5 (b) shows the case in which S_i is within the ring-encircled region, and its distance to L_k is smaller than r. In this case, the query issued by the sink is forwarded along the direction of $L_k \rightarrow S_i$ before it is intercepted by some node on the ring. The subsequent process is the same as the previous case.

Figure 5 (c) shows a complicated case, where S_i is inside the ring-encircled region, and its distance to L_k is larger than r. In this case, the query is first forwarded along the direction of $S_i \rightarrow L_k$. When the message arrives at a node whose distance to L_k is less than r, the forwarding direction of the message is changed to $L_k \rightarrow S_i$. The subsequent process is the same as the previous case.

When the storing node of a target changes, the location of the new storing node must be updated at the index nodes for the target. The process of transferring an index update message to the index ring is similar to that of transferring an index query message. When the message arrives at an index node on the ring, the node updates its index, and forwards the message along the circle in the clockwise direction. If the node is a forwarding node, it is simply sent to the counterclockwise neighbor. The message is dropped when it is forwarded back to a node that has already received it.

4.2.3 Dealing with Node Failures

To detect the failures of index nodes, neighboring nodes on an index ring should monitor each other by exchanging beacon messages periodically. In this paper, we consider the following failures:

Individual Node Failures:

Due to energy depletion or hardware faults, an individual node may fail. This kind of failures can be tolerated by the GAF protocol, in which a failed grid head is detected and replaced by other nodes in the same grid. When a new grid head is elected after the old grid head (which is on an index ring) fails, the information (e.g., some indexes and pointers) held by the old grid head is lost. However, the

new head can receive beacon messages from its neighboring nodes on the ring. From these messages, it knows that it is a node on the index ring, and can get the lost information from the neighbors.

Clustering Failures:

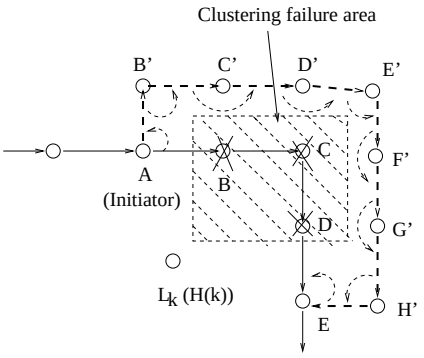


Figure 6: Dealing with clustering failures

Due to some environmental reasons, the nodes within a certain region may all fail or be isolated. This kind of failures is called *clustering failures*. Clustering failures may break an index ring. Figure 6 shows a scenario where an index ring is broken after a clustering failure occurs. In this scenario, nodes B, C and D are failed and cannot be replaced by other nodes in their grids, since all the nodes in these grids are dead. The failure can be detected by the neighbors of the failed nodes (i.e., nodes A and E), who do not receive beacon messages from the failed nodes for certain time interval. The counterclockwise neighbor (i.e., node A) of the failed nodes initiates a process to repair the ring.

In the repairing process, the initiator A sends out a $recovery$ message, and the message is forwarded around the failed region based on the *right hand rule* [15], until the message arrives at a functioning node on the ring. The process is depicted in Figure 6, and is further explained as follows: The initiator A forwards the $recovery$ message to B', which is the next neighbor of A and is sequentially counterclockwise about A from edge $\langle A, B\rangle$. On receiving the message, B' forwards the message to its functioning neighbor C', which is the next one sequentially counterclockwise about B' from edge $\langle B', A'\rangle$. The process continues, until the message arrives at E, which is a functioning node on the ring. When E receives the $recovery$ message, it consumes the message, and sends an ack message back to A along the reversed path $\langle E, H', G', F', E', D', C', B', A\rangle$. When A receives the ack message, the repairing process completes, and the broken fragment of the ring, i.e., $\langle A, B, C, D, E\rangle$,

is replaced by a new fragment $\langle A, B', C', D', E', F', G', E\rangle$.

4.2.4 Ring Reconfiguration for Load Balance

It is possible that some nodes on the index ring become overloaded compared to others. For the purpose of load balance, these nodes should be replaced by some lightly-loaded nodes. In the ARI scheme, we propose a simple algorithm to support load balance-oriented ring reconfiguration: Each node maintains a *willing flag* to indicate whether it is willing to be a node on an index ring. Initially, the flag is turned on. Each node i periodically exchanges its residual energy level (denoted as e_i) with its neighbors, and calculates the average residual energy level of all its neighbors (denoted as $\overline{e(i)}$). Node i turns off its willing flag when: $e_i < \alpha * \overline{e(i)}$, where α is a system parameter. When a node on the ring turns off its willing flag, it sends a $quit$ message to its counterclockwise neighbor. On receiving the message, the neighbor initiates a process similar to the ring repairing process (refer to 4.2.3) to remove the quitting node and reconfigure the ring.

5 Enhancements

5.1 Reduce the Overhead of Index Updating

With the ARI scheme, a source (i.e., the storing node for a target) needs to update its location at the index nodes when its location changes. If a target frequently moves, its storing node also frequently changes, and the overhead of index updating may become very large. Further, if the source is only seldom queried, most of the updates may be useless. Based on the above observations, we propose a *lazy index updating (LIU)* mechanism to reduce the overhead of index updating.

5.1.1 Basic Idea of LIU

The basic idea of LIU is illustrated in Figure 7. When a source changes, as shown in Figure 7 (a), it may not immediately update its location at the index nodes if the new location is not far away from the location registered at the index nodes. Only when the distance exceeds a certain threshold, as shown in Figure 7 (b), the source updates its location at the index nodes.

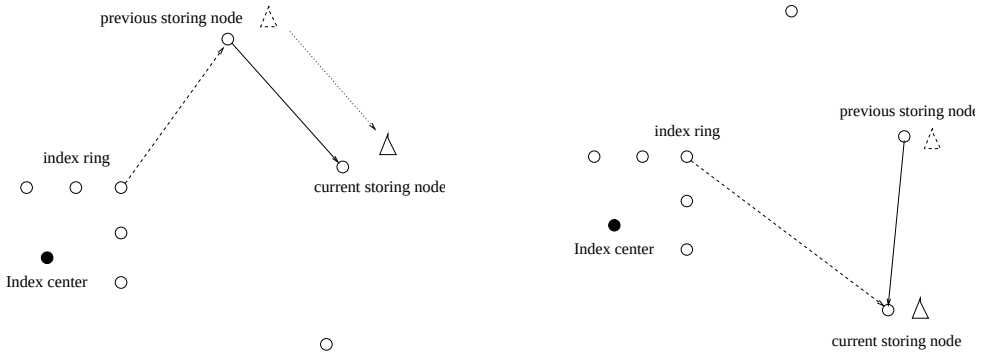


Figure 7: Lazy index updating

Since the index nodes of a source may not know the current location of the source, some measures should be taken to guarantee that a query for the source can still reach the source. For this purpose, as shown in Figure 7 (a), each old storing node temporarily keeps a pointer to the next storing node. So that, when the old storing node receives a data query, it can redirect the query to the next storing node, and the process continues until the query reaches the current storing node.

Before using the lazy index updating mechanism, we need to solve the following problems: (1) whether or not updating the source location at the index nodes when the location is changed. (2) how long an old storing node should keep a pointer to the next storing node. Next, we first study these problems in theory, and then propose a protocol to solve them in a distributed fashion.

5.1.2 Dynamically Adjust the Frequency of Index Updating

We consider a certain target, and assume that:

(1) The source of the target changes when its distance to the current detecting node of the target is greater than d_c hops.

(2) During a certain time period, the source changes at an average rate of r_c (per second), and the average cost of updating the location at the index nodes is d_u hops.

(3) During the same period, the source is queried at an average rate of r_q.

Therefore, if the original ARI scheme is used, the total cost of querying the source and updating its location at the index nodes is:

$$C = r_q \cdot (d_{i,i} + d_{i,o}) + r_c \cdot d_u \text{ hops (per second)}, \tag{1}$$

where $d_{i,i}$ is the average distance (in hop) between a sink and the closest index node, and $d_{i,o}$ denotes the average distance between the index node and the source.

When LIU is used, we assume that a source updates its location at the index nodes after every α ($\alpha \geq 1$) location changes. Thus, the total cost of querying the source and updating the index nodes becomes:

$$C' = r_q \cdot (d_{i,i} + d_{i,o} + \frac{\alpha}{2} \cdot d_c) + \frac{r_c}{\alpha} \cdot d_u \text{ hops (per second)},$$

and

$$C - C' = r_c \cdot d_u - (\frac{\alpha}{2} \cdot r_q \cdot d_c + \frac{r_c}{\alpha} \cdot d_u). \tag{2}$$

To maximize $C - C'$,

$$\alpha = max(1, \sqrt{\frac{2r_c \cdot d_u}{r_q \cdot d_c}}). \tag{3}$$

Accordingly, an old storing node should keep a pointer to the next storing node for at least $\alpha \cdot \frac{1}{r_c}$. To deal with the case that the updating rate (i.e., the source location change rate) suddenly changes a lot, the threshold may be changed to

$$\beta = \alpha \cdot \frac{1}{r_c} \cdot (1 + \rho), \text{ where } \rho > 0. \tag{4}$$

Here, β represents the time period that an old storing node should keep a pointer to the next storing node. ρ is a parameter related to the extend of updating rate changes that can be accommodated by the scheme.

5.1.3 The Protocol

A storing node can be in one of the following states: *new*, *registered*, *old*, and *terminated*. The transitions between these states are illustrated in Figure 8, and is further explained as follows.

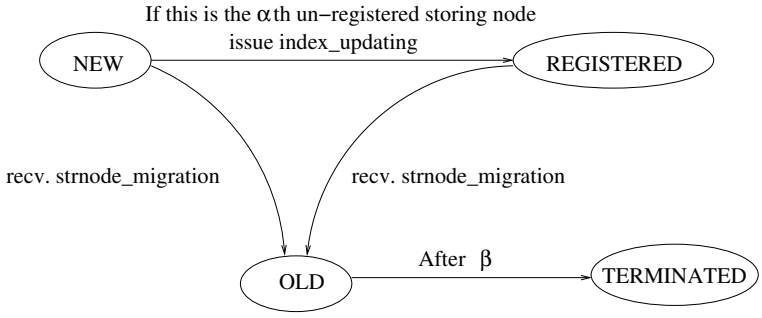


Figure 8: The state transition of a storing node

When a detecting node selects node N_s as a new storing node, N_s sets its state as *new*, and sends a message $strnode\_migration$ to the previous storing node. N_s will also receive following information from the previous storing node: (1) the empirical query rate: $\hat{r_q}$; (2) the empirical rate of changing storing node: $\hat{r_c}$; (3) the empirical overhead of updating the storing node location at the index nodes: $\hat{d_u}$; and (4) the number of un-registered storing nodes (i.e., whose locations are not registered at the index nodes) since the last registered one: n_u. Based on the above information, N_s computes the empirical value of α, i.e.,

$$\hat{\alpha} = max(1, \sqrt{\frac{2\hat{r_c} \cdot \hat{d_u}}{\hat{r_q} \cdot d_c}}).$$

If $n_u = \hat{\alpha} - 1$, N_s should send out a message $index\_updating$ to the index nodes to register its location and change its state to *registered*. Otherwise, it will not change its state. When a new storing node is selected, as shown in Figure 8, N_s will receive a message $strnode\_migration$, and changes its state to *old*. An old storing node keeps a pointer to the next storing node for a time period of length

$$\hat{\beta} = \hat{\alpha} \cdot \frac{1}{r_c} \cdot (1 + \rho).$$

During the time period that N_s is a storing node, N_s continuously monitors the queries and the storing node changes, and the empirical values accordingly. For example, after receiving a query, N_s adapts $\hat{r_q}$ as follows:

$$\hat{r_q} = \frac{1}{2} \cdot \hat{r_q} + \frac{1}{2} \cdot \frac{1}{\tau},$$

where τ is the time interval between the current query and the last query. Similarly, $\hat{r_c}$ is adapted when receiving a $strnode\_migrate$, and $\hat{d_u}$ is adapted when an index updating finishes.

5.2 Reduce the Query Overhead

A sink may continuously monitor a moving target by frequently querying the current storing node of the target. With ARI, the sink needs to query the index nodes before querying the source. Therefore, the overhead for query can be high, particularly when the sink and the source are far away from the index nodes. To reduce the overhead, we propose a lazy index query (LIQ) mechanism, which enhances the lazy index updating mechanism as follows:

- An old storing node keeps a pointer to the next storing node for at least $max(\beta, \theta)$, where θ is system parameter.
- When a source replies a data message to a sink, it attaches its location to the message. On receiving the message, the sink caches the location.
- When a sink wants to query the source of a target, it first checks if it has cached the location of the source. If the location is cached and the caching time is less than θ, it will send query directly to the source. Otherwise, the query is sent to the index nodes.

6 Performance Evaluations

6.1 Simulation Model

We develop a simulator based on ns2 (version 2.1b8a) [1], to evaluate and compare the index-based data dissemination scheme (with ARI) to three data dissemination schemes: ES, DCS [20] and LS (with source-initiated sink-source matching mechanism) [29]. In this simulator, the MAC protocol is based on IEEE 802.11[1] and the transmission range of each node is $40m$ [14]. 2500 sensor nodes are distributed over a $850 \times 850m^2$ flat field. Therefore, each node has about 17.4 neighbors in average. The field is divided into $17 \times 17m^2$ GAF grids and one sensor node is randomly deployed in each grid. This guarantees that any grid has one node and the nodes in any two neighboring grids are within the communication range of each other since their distance is at most $17m \times \sqrt{5} < 40m$.

[1] In the simulation, packet transmission follows the RTS/CTS/ACK protocol. But we do not follow the format of IEEE 802.11. We simply let the packet size be 10 bytes when an update message or a query message is transmitted; and we let the packet size be 50 bytes when a data message is transmitted. This is also shown in Table 2.

Once the sensor nodes are deployed, the network topology is fixed in the simulations, so that all the peformance comparisons are based on the same topology. Note that we have also veried the network topology (with the same restriction that each grid has exactly one node), and the simulation results do not change much.

Several targets (for simplicity, each target has a unique type; the number of target types is denoted as N_t) are deployed in the detecting region. Each target may move in any direction and its average velocity is denoted as v. When a target enters a grid, the sensor node located in that grid can detect it. If the index-based data dissemination scheme is simulated, the sensor node that first detects a target becomes the initial source (storing node) of the target. When the distance between the current detecting node of a target and the source is larger than a threshold, the source is changed to be the current detecting node. Based on our assumptions described in Section 2.2, each type of target has a name, which is an integer between 0 and $N_t - 1$. We use a simple hash function to map the name of a target to one of N_i index centers, which are uniformly distributed in the detecting field. The mapping rule is as follows: $H(k) = L_{k \bmod N_i}$, where k is the name of a target and L_j is the location of the j^{th} index center. Any sensor node can be a sink which issues a query for the data of a certain target. Table 2 lists most of the simulation parameters.

6.2 Simulation Results

The simulation includes three parts: The first part (Section 6.2.1) evaluates and compares the overhead of four data dissemination schemes to verify the effectiveness of the analytical results presented in Section 3.2. The second part (Section 6.2.2-6.2.4) evaluates the ARI scheme to show that this scheme does not have high overhead and is resilient to clustering failures and can achieve load balance. The third part (Section 6.2.5-6.2.6) evaluates the proposed enhancements to the ARI scheme.

In our simulation, the sensor nodes are uniformly distributed in the field. The movement pattern of the targets and the query pattern can be random. However, if the movement pattern varies between experiments, the measured message complexity and traffic can be much different between any two simulation runs. Therefore, in each experiment, we fix the target movement trace, but allow the

Table 2: Simulation Parameters

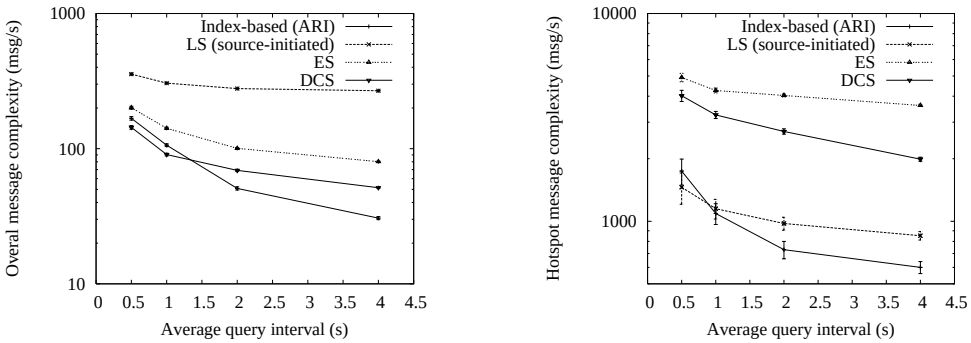
Parameter	Value
field size (m^2)	850×850
number of nodes	2500
communication range (m)	40.0
grid side (m)	17.0
number of target types: N_t	10
data update rate: r_d (per target per second)	0.25
number of index centers: N_i	4
the migration threshold for a source (m):	34.0
initial radius of an index ring: r (m)	34.0
initial number of index nodes on a ring: m	4
simulation time for each experiment (s)	1000.0
average velocity of a mobile target: v (m/s)	1.0-6.0
size of an update message ($byte$)	10
size of a query message ($byte$)	10
size of a data message ($byte$)	50

queries to be randomly generated. Figures 9-17 show the result of our simulation. In these figures, each data point is the mean of ten simulation runs. The standard variances are also shown.

6.2.1 Comparing the performance of data dissemination schemes

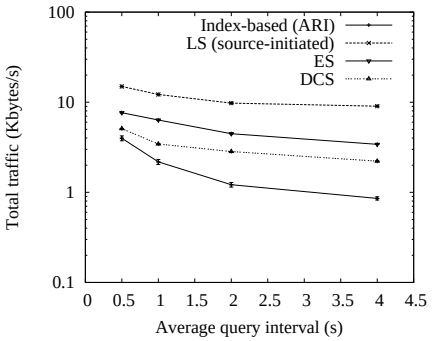
We first evaluate and compare the overall message complexity of the data dissemination schemes, and the results are shown in Figure 9 (a). LS is shown to have the highest message complexity, since it needs to flood control messages to the whole network. ES and DCS have the same order of message complexity in theory. However, ES pushes data to a base station outside of the network, and hence has larger overhead for pushing and retrieving data than DCS, which pushes data to nodes within the network. Figure 9 (a) also shows that the index-based scheme and DCS has similar overall message complexity. When the query interval is small (i.e., the query rate is high), the index-based scheme has a little bit higher message complexity than DCS. The trend is reversed when the query interval is large (i.e., the query rate is low). This phenomenon is consistent with *Observation 1* presented in Section 3.2.

Figure 9 (b) shows the hotspot message complexity of the data dissemination schemes. ES is shown to have the highest hotspot message complexity, since the hotspot nodes (i.e., the edge nodes close to

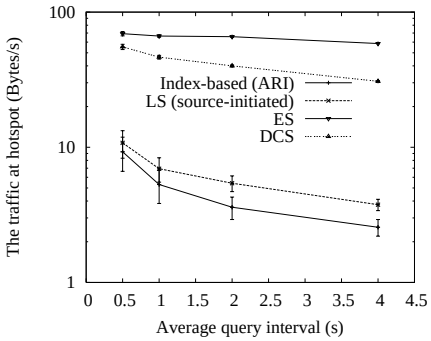


(a) Overall message complexity

(b) Hotspot message complexity



(c) Overall traffic



(d) Hotspot traffic

Figure 9: Evaluating data dissemination schemes ($v = 1.0m/s$)

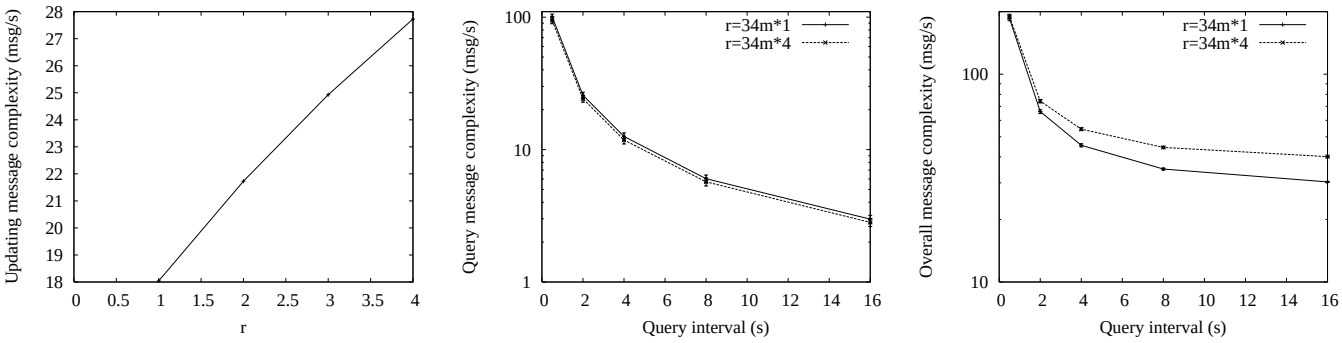
the base station) need to forward all the messages related to storing and retrieving data, which include the data pushed from the detecting nodes, the queries from the sinks, and the responses sent from the base station. DCS has much smaller hotspot message complexity, because a hotspot node (i.e. a node near a index center) only processes messages related to one type of target whose data are stored there. LS is shown to have smaller hotspot message complexity than DCS. This is due to the fact that, in our simulations, the target does not move quickly and the number of index centers is small (i.e., 4). According to the theoretic estimation results (4.2) and (3.2) presented in Section 3.2, the index-based scheme has a smaller number of messages at hotspots than DCS. From Figure 9 (b), we can also see that, the index-based scheme has lower hotspot message complexity than LS only when the query interval is large (i.e., the query rate is small), which is also consistent with *Observation 2* presented in Section 3.2.

The overall traffic of the data dissemination schemes are shown in Figure 9 (c). From the figure, we can see that LS has very large traffic due to the fact that it floods control messages over the whole

network. ES and DCS also cause large overall traffic, since they both push data from detecting nodes to storing nodes regardless of queries. ES has larger traffic than DCS, because it pushes data to nodes outside of the network, and hence brings more communication overhead. The index-based scheme has the smallest traffic among all the schemes. This is due to the fact that, in this scheme, data are sent from a source to a sink only when the data is queried, and the control messages (i.e., queries and index updates) are never flooded.

Figure 9 (d) shows the hotspot traffic of the data dissemination schemes. From this figure, we can see that ES has the highest hotspot traffic, which is followed by DCS. The index-based scheme has the smallest hotspot traffic when the query interval is large (i.e., the query rate is high). The phenomenon is similar to that shown in Figure 9 (b) due to the same reasons as explained before.

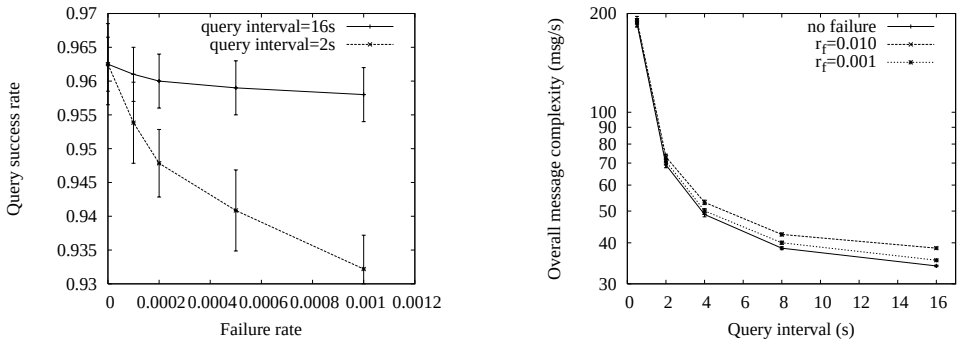
6.2.2 Impact of Parameter r on The Performance of ARI



(a) (b) (c)

Figure 10: Impact of r ($v = 3.0m/s, m = 8$)

Figure 10 (a) shows that the index updating overhead increases as r increases. This is because, each index updating message needs to travel the whole index ring. As r increases, the index ring also increases, and an index updating message has to travel a longer path. Figure 10 (b) shows that increasing r can reduce the overhead of querying, where the query message complexity refers to the total number of hops that all the query messages must be forwarded. This is due to the fact that most sinks are outside of the index ring. As r increases (i.e., the ring expands), the query message can be intercepted by an index node soon. Since the reduction in querying overhead is smaller than the



(a) (b)

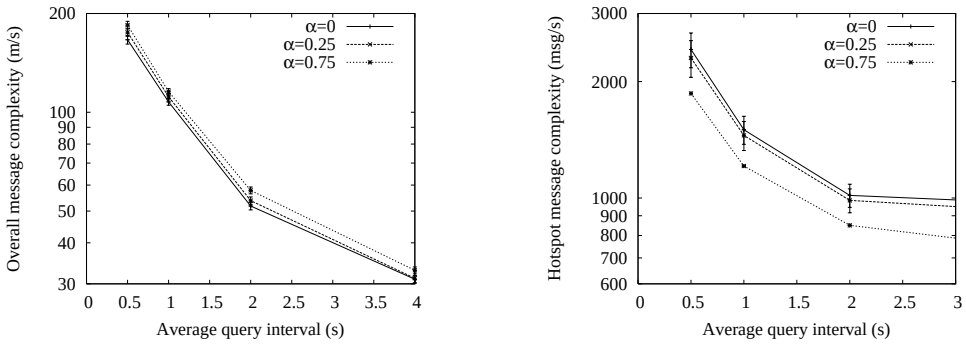
Figure 11: Evaluating the fault tolerance feature of ARI ($r = 34m * 2, m = 8, v = 3.0m/s$)

increase in updating overhead, as shown in Figure 10 (c), the overall message complexity increases as r increases. However, the total message complexity does not increase very much if r is not large.

6.2.3 Tolerating Clustering Failures

We now show how the ARI scheme reacts to clustering failures. Similar to [20], the *up/down* model is adopted in the simulations: each $17 * 17m^2$ grid may be isolated or fail at an average rate denoted as r_f, and an isolated or failed grid may recover at an average rate denoted as r_c (We assume $r_c = 3r_f$). The success rate of the queries and the overall message complexity are measured as r_f (r_c) varies, and the results are shown in Figure 11.

From Figure 11 (a), we can see that the query success rate decreases slightly when r_f increases. This is due to the fact that, when the index ring is broken due to clustering failure, the ARI scheme guarantees that a query can still be routed to a normal node on the index ring, and routed to one functioning index node. Only when all index nodes are failed, or a ring node fails after it has received a query and before it has not forwarded the query, the query is dropped. Figure 11 (a) also shows that, when query interval decreases (i.e., the query rate increases), the query success rate drops more quickly as r_f increases. This is because, the frequency that a ring node fails after receiving a query increases as the query rate increases, which results in more query messages being dropped. Note that, the success rate is smaller than 1.0 even when the failure rate is 0 since a query message may be dropped due to traffic congestion.



(a) (b)

Figure 12: Evaluating the load balance feature of ARI ($r = 34m * 2, m = 8, v = 1.0m/s$)

Figure 11 (b) shows that the overall message complexity is slightly increased as the failure rate increases. This can be explained as follows: When an index ring is broken due to clustering failures, the ring should be repaired, and hence some communication overhead is introduced. Also, the repaired ring becomes longer, which increases the overhead for index updates.

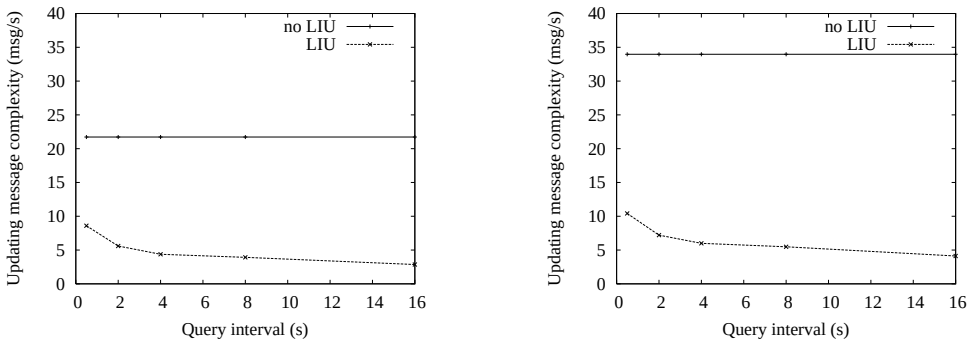
6.2.4 Load Balance

Next, we evaluate the ARI scheme's ability to achieve load balance. In this simulation, each node is initially equipped with an energy of $0.2J$, and other simulation parameters are the same as the previous simulations. We measure the overall message complexity and the hotspot message complexity, when the load balance module is turned on (i.e., the load balance parameter is larger than 0) or off (i.e., the load balance parameter is set to 0).

Figure 12 (a) shows the overall message complexity. We can see that, the overall message complexity is increased by about 10% as α changes from 0 to 0.75. This is due to the reason that, the index ring has to be dynamically reconfigured when the load balance module works. With the dynamic reconfigurations, as shown in Figure 12 (b), the hotspot message complexity is decreased by around 25%.

6.2.5 Evaluating LIU

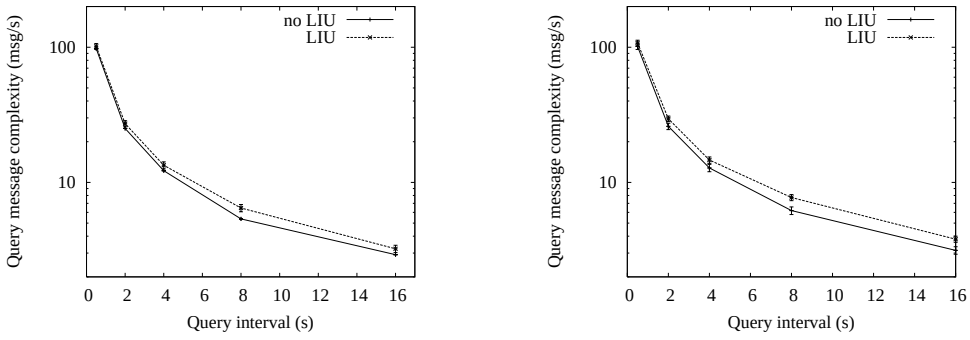
Figure 13 shows that LIU can significantly reduce the overhead of index updating. This is due to the following reasons: In the original ARI scheme (without LIU), a source needs to update its new loca-



(a) v=3.0m/s (b) v=6.0m/s

Figure 13: The index updating message complexity with/without LIU ($r = 34m * 2, m = 8$)

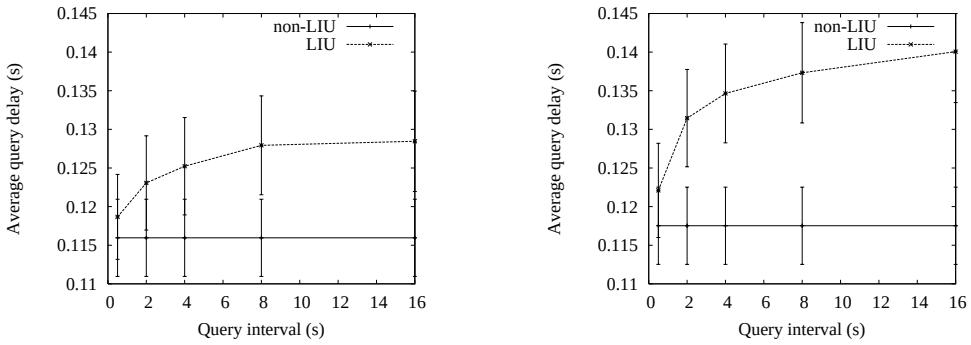
tion at the index nodes whenever its location changes. This may introduce large overhead. However, when LIU is used, a source can dynamically adjust the index updating frequency based on several factors (e.g., the query rate, the frequency of changing its source location, *etc.*) to minimize the index updating overhead. The figure also shows that the overhead decreases as the query interval increases. This is because, as shown in Eq. (3), α is proportional to $\sqrt{\frac{1}{r_q}}$ (r_q is the query rate). So, α increases as the query interval increases (i.e., r_q decreases). As a result, the frequency and the overhead of index updating are reduced. Comparing Figure 13 (a), (b) and (c), we can further find that LIU reduces more overhead as the target velocity (v) increases. This is because, when ARI is used, the frequency and the overhead of index updating increase rapidly as v increases. However, LIU can slow down the increase by adjusting the index updating frequency (i.e., increasing α) according to Eq. (3).



(a) v=3.0m/s (b) v=6.0m/s

Figure 14: The query message complexity with/without LIU ($r = 34m * 2, m = 8$)

Figure 14 and 15 show that LIU increases the query overhead in terms of query message complex-

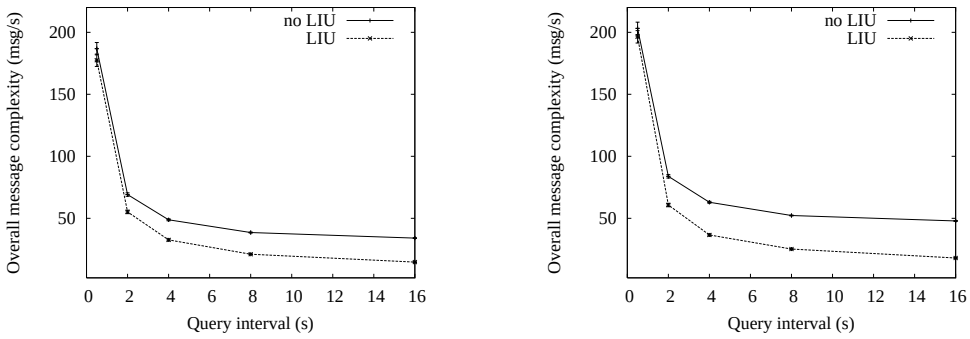


(a) v=3.0m/s (b) v=6.0m/s

Figure 15: The average query delay with/without LIU ($r = 34m * 2, m = 8$)

ity and query delay. This phenomenon can be explained as follows. When LIU is not used, the index nodes know the current source locations, and a query message travels a path: $sink \rightarrow index\_node \rightarrow current\_source$. However, when LIU is used, the index nodes may not know the current source locations, and a query message may travel a longer path: $sink \rightarrow index\_node \rightarrow old\_source_1 \rightarrow \cdots old\_source_m \rightarrow current\_source$. A longer querying path results in larger querying overhead. The figures also show that the overhead increases as the query interval increases. This is because, as shown in Eq. (3), the index updating frequency (α) increases as the query rate r_q decreases (i.e. the query interval increases). Consequently, the source location recorded by the index nodes becomes further older, and a query message may travel a longer path as the query interval increases. Comparing Figure 14 (15) (a), (b) and (c), we can find that the query overhead increases as the target velocity (v) increases, since the index updating frequency increases more slowly than the frequency of changing source location, which increases the length of querying paths and the querying overhead.

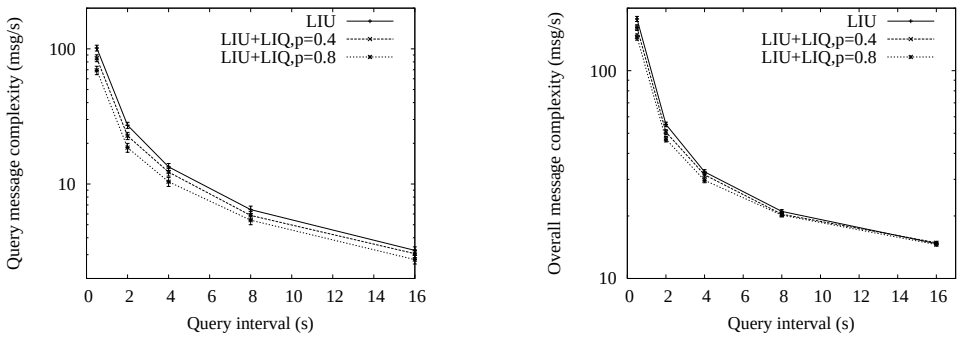
Figure 16 shows that LIU reduces the total message complexity. The phenomenon can be explained based on Figure 13 and Figure 14. LIU significantly reduces the overhead of index updating, and at the same time, it slightly increases the querying overhead. As a result, the total message complexity is reduced. Furthermore, as the target velocity (v) increases, the savings of index updating increase more rapidly than the increase of query overhead. Consequently, more total message complexity is reduced as v increases.



(a) v=3.0m/s (b) v=6.0m/s

Figure 16: The total message complexity with/without LIU ($r = 34m * 2, m = 8$)

6.2.6 Evaluating LIQ



(a) (b)

Figure 17: The message complexity with/without LIQ ($r = 34m * 2, m = 8, v = 3.0m/s$)

To evaluate LIQ, we randomly select a set of sinks, each continuously query the same source. The total number of sink is 50, and the number of the selected sinks is $50p$ (p varies from 0.1 to 0.8). Also, a cached source location is valid if and only if it has been cached for less than $100s$ (i.e., the parameter θ of the LIQ scheme is set to $100s$).

Figure 17 shows that LIQ can reduce the query message complexity and the overall message complexity. This is due to the following reasons: With LIU, each query message should travel a path: $sink \rightarrow index\_node \rightarrow old\_source_1 \cdots \rightarrow old\_source_m \rightarrow current\_source$. When LIQ is used, a sink may use the source location information cached locally to locate the source and avoid accessing the index nodes. Consequently, the querying path becomes: $sink \rightarrow old\_source_1 \cdots \rightarrow old\_source_m \rightarrow current\_source$. The reduction in querying overhead become more significant if

the sink and the source is close to each other, and they are both far away from the index nodes. Figure 17 also shows that the benefit of using LIQ becomes larger as p increases or the query interval reduces. This phenomena can be explained as follows: As p increases, a larger portion of queries are issued from the same sink. Once a sink has queried a source, the source location information is cached locally, and can be used for the next query as long as the information is still valid. Also, as the query interval reduces, the cached source locations can be updated more frequently. Consequently, the cache hit ratio is reduced and hence the querying overhead decreases.

7 Related Work

In this section, we review some prior work on data dissemination and storage in sensor networks.

Early proposals on data dissemination schemes are mainly based on external storage or local storage. In LEACH [13], sensor nodes send data to an external base station through a local cluster head immediately after the data are generated. TAG [17] is more advanced form of ES scheme which incorporates some in-network aggregation of data. The directed diffusion [14] and TTDD [29], described in the early part of this paper, are example of local storage-based schemes.

Our ARI scheme is closely related to data-centric storage schemes, which include GHT [20], DIMENSIONS [9], DIFS [11], DIM [16] and GEM [18]. In GHT, data of an event is stored to a location determined by the name of the event. GEM follows the similar idea but does not depend on geographic information. In GEM, nodes are assigned labels based on a embedded graph for the sensor network, and a mapping can be made between the labels and the locations for storing data of certain events. Different from GHT and GEM, in ARI, data are stored locally and only the indexes to the data are stored based on their related events. By this, unnecessary transfers of data can be avoided, which makes ARI efficient for scenarios that a large volume of data are generated but only a small fraction of them is queried. DIFS and DIM use the same set of primitives as GHT, but they extend the data-centric storage approach to provide spatially distributed hierarchies of indexes to data. ARI differs from these schemes in that it is used for indexing events based on their type.

8 Conclusions and Future Work

In this paper, we proposed an index-based data dissemination scheme with adaptive ring-based index (ARI). This scheme is based on the idea that sensing data are collected, processed and stored at the nodes close to the detecting nodes, and the location information of these storing nodes is pushed to some index nodes. The index nodes for the targets of one type forms a ring surrounding the location which is determined based on the event type, and the ring can be dynamically reconfigured for fault tolerance and load balance. We also proposed several mechanisms to optimize the ARI scheme. Analysis and simulations were conducted to evaluate the performance of the proposed index-based scheme. The results show that the index-based scheme outperforms the ES scheme, the DCS scheme, and the LS scheme. The results also show that using the ARI scheme can tolerate clustering failures and achieve load balance, and the proposed optimization mechanisms can further improve the system performance.

Finally, we highlight some directions for our future work on the ARI scheme: We first plan to implement our ARI scheme in a real sensor network testbed (e.g., MICA motes/TinyOS) to study more realistic issues of the proposed schemes. One important topic for further investigation is the complexity for combining ARI with GAF and GPSR. Although GAF and GPSR are both lightweight protocols, combining them with ARI may raise new issues. For example, query or registration messages may reach a grid that is performing a grid head re-election. This may result in message congestion and even message dropping. The rotation of grid heads also complicates the maintenance of index information, and demands for a mechanism to coordinate related index nodes for index querying and updating. The ARI scheme should also be further enhanced to address other factors such as errors in localization and the existence of obstacles in the network.

As another important direction, we will study how to improve the efficiency of using storage space in the sensor network. This is essentially important when the network is expected to work for a long time and thus the generated data may use up the space affordable by the network. Some data compression and aging techniques, similar to those proposed in [8], should be developed to address

the problem.

Acknowledgments

We would like to thank the editor and the anonymous referees whose insightful comments helped us to improve the presentation of the paper. A preliminary version of the paper appeared in IEEE International Conference on Network Protocols (ICNP), 2003. This work was supported in part by the US National Science Foundation under grants CAREER CCR-009277 and ITR-029711.

References

[1] The CMU Monarch Project Wireless and Mobility Extensions to ns. *http://www.monarch.cs.cmu.edu/cmu-ns.html*, October 1999.

[2] US Naval Observatory (USNO) GPS operations. *http://tycho.usno.navy.mil/gps.html*, April 2001.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, March 2002.

[4] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in Ad Hoc wireless networks. *Wireless Networks*, 7(6):609–616, November 2001.

[5] R. Brooks, P. Ramanathan, and A. Sayeed. Distributed target classification and tracking in sensor networks. *Proceedings of the IEEE*, 91(8):1163–1171, 2003.

[6] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor location for very small devices. *IEEE Personal Communication, Special Issue on "Smart Space and Environments"*, 7(5):28–34, October 2000.

[7] A. Cerpa, J. Elson, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. *The Proceeding of the First ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[8] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. pages 89–102, 2003.

[9] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new Data Handling architecture for Sensor Networks? *Proceedings of the ACM Workshop on Hot Topics in Networks*, pages 143–148, 2002.

[10] A. Ghose, J. Grossklags and J. Chuang. Resilient data-centric storage in wireless ad-hoc sensor networks. *Proceedings the 4th International Conference on Mobile Data Management (MDM'03)*, pages 45–62, 2003.

[11] B. Greenstein, S. Ratnasamy, S. Shenker, R. Govindan, and D. Estrin. DIFS: a distributed index for features in sensor networks. *Ad Hoc Networks*, 1(2-3):333–349, 2003.

[12] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J. Stankovic, T. Abdelzaher, B. Krogh. Lightweight detection and classification for wireless sensor networks in realistic environments.

Proceedings of the 3rd international conference on Embedded networked sensor systems (Sensys), pages 205–217, 2005.

[13] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocal for wireless microsensor network. *Proc. of the Hawaii International Conference on System Sciences*, January 2000.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication. *Proceedings of the Sixth Aunual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 56–67, August 2000.

[15] B. Karp and H. Kung. GPSR: greedy perimeter stateless routing for wireless networks. *The Sixth Aunual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, pages 243–254, August 2000.

[16] X. Li, Y. Kim, R. Govindan, W. Hong. Multi-dimensional range queries in sensor networks. *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 63–75, 2003.

[17] S. Madden, M. Franklin, J. Hellerstein and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[18] J. Newsome and D. Song. Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information. pages 76–88, 2003.

[19] S. Patterm, S. Poduri, and B. Krishnamaachari. Energy-quality tradeoffs for target tracking in wireless sensor networks. *Proceedings of the 2nd Internal Conference on Information Processing in Sensor Networks (IPSN)*, pages 32–46, 2003.

[20] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mob. Netw. Appl.*, 8(4):427–442, 2003.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *Proceeding of ACM SIGCOMM'01*, pages 14–29, August 2001.

[22] M. Roussopoulos and M. Baker. CUP: controlled update propagation in peer-to-peer networks. *Proceedings of the 2003 USENIX Annual Technical Conference*, June 2003.

[23] A. Rowstron and P. Druschel. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. *The 18th IFIP/ACM Symposium on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, August 2001.

[24] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. *Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (Sensys)*, pages 1–12, 2004.

[25] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for Internet applications. *Proceedings of ACM SIGCOMM'01*, pages 149–160, August 2001.

[26] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson and D. Culler. An analysis of a large scale habitat monitoring application. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 214–226, 2004.

[27] Q. Wang, W. Chen, R. Zheng, K. Lee, and L. Sha. Acoustic target tracking using tiny wireless sensor devices. *Proceedings of the 2nd Internal Conference on Information Processing in Sensor Networks (IPSN)*, pages 642–657, 2003.

[28] Y. Xu, J. Heidemann and D. Estrin. Geography-informed energy conservation for ad hoc routing. *Proceedings of ACM MOBICOM'01*, pages 108–116, July 2001.

[29] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang. A two-tier data dissemination model for large-scale wireless sensor networks. *ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pages 148–159, September 2002.

[30] W. Zhang, G. Cao and T. La Porta. Data dissemination with adaptive ring-based index for wireless sensor networks. *Proceedings of the IEEE International Conference on Network Protocols'03*, pages 305–314, November 2003.

[31] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: an infrastructure for faul-tolerant wide-area location and routing. *Technical Report UCB/CSD-01-1141*, 2001.

[32] F. Zhao, J. Liu, L. Guibas, and J. Reich. Collaborative signal and information processing: An information directed approach. *Proceedings of the IEEE*, 91(8):1199–1209.