This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

A Distributed Spatial-Temporal Similarity Data Storage Scheme in Wireless Sensor Networks

Haiying Shen, Lianyu Zhao and Ze Li
Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29634
Email: {shenh, lianyuz, zel}@clemson.edu

Abstract—Since centralized data storage and search schemes often lead to high overhead and latency, distributed data centric storage becomes a preferable approach in large-scale wireless sensor networks (WSNs). However, most of existing distributed methods lack optimization for spatial-temporal search to query events occurred in a certain geographical area and a certain time period. Furthermore, for data search routing, most methods rely on locating systems (e.g., GPS), which consume high energy. This paper proposes a distributed spatial-temporal Similarity Data Storage scheme (SDS). SDS provides efficient spatial-temporal and similarity data searching service, and is applicable for both static and dynamic WSNs. It disseminates event data in such a way that the distance between WSN neighborhoods represents the similarity of data stored in them. In addition, SDS carpooling routing algorithm efficiently routes messages without the aid of GPS. Theoretical and experimental results show that SDS yields significant improvements on the efficiency of data querying compared with existing approaches, and obtains stable performance in dynamic environments.

Index Terms—Wireless sensor networks, Data centric storage, Similarity search.

1 Introduction

A wireless sensor network (WSN) is a wireless network consisting of a large number of distributed low-power and inexpensive sensor devices. WSNs can be used for monitoring the environment and observing certain phenomena. Particularly, WSNs have been used in many military and civilian application areas such as military target tracking, habitat monitoring, health monitoring and industrial process control [1].

A data storage scheme in a WSN offers data storage and search services for sensed events. One challenge faced by data storage schemes is to efficiently aggregate and query data in the network. Sensor nodes are battery operated. Thus, energy is one of the major constraints in accessing the data captured by the WSN nodes. Also, fast data searching is another requisite for a data storage scheme, especially in time-critical applications.

The second challenge is similarity search for multi-attribute data. Similarity search enables users to search data within a specified similarity range to a query in addition to the exact matched data. For example, $5085|F-16|AirForce|Division2|US$ may also be interesting data for a requester with the query $4957|F-16|AirForce|Division1|US$. Similarity search is also needed in many applications such as finding similar flow patterns in ocean concurrent monitoring and wildlife activity patterns in habitat monitoring. Also, it is often more appropriate for a user to formulate search requests in less precise terms, rather than defining a sharp limit.

The third challenge is spatial-temporal search, which contributes to real-time data retrieval. Spatial-temporal search allows users to query the data of events that occurred at a specified physical location within a certain time period. For example, for the query "how many pedestrians are there in the geographical region X during 7:00pm-8:00pm, January 4, 2010?" or "tell me in which direction the vehicle in region Y is moving?", if a user receives data for the entire area covered by a large-scale WSN during all the time, the latency to process the received data may lead to disastrous delay in time-critical military operations.

Previous data retrieval solutions in WSNs can be classified into three approaches [2], [3]: *external storage*, *local storage* and *data-centric storage*. In *external storage* [4], [5], [6], [7], [8], data is sent to the base station without waiting for a user to send a query and nodes send queries to the base station for data searching. This may waste energy when data that is of no use by users is sent to the base station, thus resulting in unnecessary communication costs and bottlenecks. Also, since the base station is solely responsible for data storage and responding, it easily becomes a bottleneck, which may result in delayed service. External storage approaches may also lead to unbalanced energy consumption because of different distances between nodes and base stations. In *local storage* [9], [10], [11], [12], [13], each node keeps the data it senses locally and uses flooding for data retrieval, which consumes a significant amount resources. *Data-centric storage* [14], [15], [16], [17], [18] schemes hash event data to locations according to data names. All data with the same general name (e.g., moving vehicle) will be stored at the node closest to the geographical location. Data queries with a particular name can then be sent

Digital Object Indentifier 10.1109/TMC.2010.214

directly to the node storing that named data, thereby avoiding flooding. This approach has been shown to be an energy-efficient data dissemination method for WSNs [9]. However, the queries for the data are directly forwarded to the locations by geographical routing [14], [16], [19]. Most geographical routing methods rely on locating systems [15] (e.g., GPS), which consumes high energy.

In spite of the efforts to develop data storage systems in WSNs, there has been very little research devoted to tackling both similarity and spatial-temporal searches. In addition, existing data storage systems are not sufficiently energy-efficient nor adequately fast. Efficiently achieving the functionalities of spatial-temporal and similarity search with low energy consumption still remains a crucial problem in WSNs. This paper proposes a distributed spatial-temporal similarity data storage scheme (SDS). In addition to offering the spatial-temporal and similarity search functionalities, SDS accelerates querying speed and reduces communication energy consumption and overhead. In SDS, nodes in a WSN are partitioned to a number of neighborhoods. Similar data is stored in one neighborhood or nearby neighborhoods. SDS preserves the similarity between data, i.e., the physical distances between neighborhoods represent the similarity between data stored in the neighborhoods. Within a neighborhood, data is further classified based on time and location. Compared to other distributed data storage schemes, SDS is advanced in that it optimizes data querying based on not only data name but also data similarity. Also, it offers spatial-temporal data searching. In addition, SDS does not need GPS to locate the positions of nodes for routing. It uses limited geographical information to achieve comparable efficiency to GPS-based geographical routing. SDS also incorporates a carpooling routing algorithm that combines the messages targeting the same destination in routing in order to improve routing efficiency.

2 Related Work

Data storage approaches in WSNs can be classified into three categories [2], [3]: external storage, local storage and data-centric storage.

External storage. External storage schemes [4], [5], [6], [7], [8] store and access all of the data generated by sensor nodes to a single sink that is located outside of the WSN. A data query needs to travel to the sink to find the data source, thus the centralized sink may become a bottleneck. This method may also waste energy when non-essential data is sent to the base station [8]. In addition, the energy cost is not distributed in balance. Because of the energy consumption, external storage schemes would only be used in small-scale WSNs with low data generation rates.

Local storage. In local storage schemes [9], [10], [11], [12], [13], all sensed data is stored locally at the sensor nodes that detected the data. Because data can reside anywhere in the network, a data query must be flooded to all sensor nodes in the network, leading to high overhead and energy cost. Directed diffusion protocol [9] was developed to save the cost in data querying. In this protocol, nodes disseminate the messages of their interests throughout the sensor network. The events matching an interest flow towards the interested nodes along multiple paths. Finally, an interested node determines routes for future data flow by choosing a high-quality path from each source node.

Zhang et al. [10] proposed an index-based data dissemination scheme in which sensed data is stored at the detecting or nearby nodes and the data location information is pushed to a number of index nodes. This scheme avoids unnecessary data transmission and controls message flooding over the entire network. TAG [11] provides SQL-like semantics in data querying. It uses a delivery tree to distribute operators such as “count”, “min” or “max” to tree nodes, and uses the aggregation method to gather querying results from the leaves to the root. GRAB [12] forwards data along interleaved mesh to a receiver. It also controls bandwidth by using the credits of data messages, thus allowing the sender to adjust the reliability of data delivery. TTDD [13] is a two-tier data dissemination protocol that maintains a grid structure with each grid having a grid server. Compared to GRAB which needs to disseminate the existence of sinks to all nodes in a mobile environment, TTDD only needs sinks to send their existence to grid servers.

Data-centric storage. Most recent research focuses on distributed data storage schemes for WSNs [14], [16], [17], [18]. These schemes map data to geographical locations and all the data with the same general name is stored at the node closest to the geographical location. Queries for data with a particular name can then be sent directly to the node storing the named data, thereby avoiding flooding. These techniques differ in the aggregation mechanisms used, but are loosely based on the idea of geographic hashing. One such data storage scheme is Geographic Hash Table (GHT) that provides a hash function for mapping event data to locations [14]. GHT hashes a data name to a key first and then decides where the data should be stored based on the key. GHT saves the data having the same name at the same location and uses geographical routing for locating data.

Distributed Index of Features (DIFS) [19] and Distributed Multi-dimensional Range Queries (DIM) [16] extend the GHT approach to provide distributed hierarchies of indices to data. In these two techniques, the storage atom is multi-attribute data. DIMENSIONS [17] incorporates long-term storage which progressively discards old data while preserving its key features for future data mining.

To improve routing performance, GLS [20] arranges each mobile node to periodically update a small set of location servers with its current location. When a node queries for the locations of other nodes, it uses predefined identifiers and a spatial hierarchy to find

a location server for those nodes. GEM [21] embeds a labelled graph to the topology of a network to enable nodes to perform efficient routing by merely knowing the labels of its neighbors. Caruso *et al.* [22] proposed a GPS-free coordinate-based routing algorithm for WSNs. TSAR [23] has two tiers: a proxy tier and a sensor tier. At the proxy tier, it uses a multi-resolution index structure. At the sensor tier, it supports adaptive summarization that trades off energy cost against overhead.

Some effort has been devoted to building practical data centric storage (DCS) structures. Since point-to-point DCS is difficult to deploy, pathDCS [24] was proposed to provide an approach that only needs the construction of a standard tree and uses tree-based communication primitives. Aly *et al.* [25] noticed the importance of uniform data distribution, and proposed KDDCS to use a K-D tree to maintain balanced data storage and avoid bottlenecks. Bian *et al.* [26] proposed a scalable routing system which is featured by the use of hierarchical location names such as country, state, and city instead of location coordinates. Thus, an aggregated routing table can be built to ensure its scalability. EASE [27] is aimed at answering approximate node location queries and reducing the overhead. It stores data for imprecise locations at certain nodes to avoid frequent location updates when an object is within the approximation radius. Rendered Path (REP) [28] has been proposed to meet the heterogeneous node deployment. REP removes the need for powerful nodes by estimating the distances between node pairs instead of directly measuring distances.

3 The Design of SDS

3.1 Design Goals and Strategies

SDS is unique in its spatial-temporal and similarity search functionalities. Meanwhile, SDS aims to reduce overhead, energy consumption and searching latency.

(1) Similarity searching functionality. This functionality is very useful for collecting data that has inner relationship, especially when a user is unsure of the exact keywords to be searched or when it is difficult to normalize attributes. SDS divides an entire monitored area into a number of grid zones, and relies on locality sensitive hashing [29] to map data to zones in a locality-preserving manner as shown in Figure 1. That is, the physical closeness of zones represents the similarity of data in the zones. Thus, by searching a particular zone's nearby zones, a node can find similar data to the data in that particular zone.

(2) Spatial-temporal searching functionality. This functionality makes it possible to search for data of events that occurred in a specified location within a specific time period. SDS tackles this challenge by building a two-dimensional space in each zone, in which it maps data to the nodes in a zone based on both location and time.

(3) Low overhead. Low overhead or energy consumption is crucial to WSNs, which are constrained by their limited energy supply. SDS always produces a small number of messages for data searching. In addition, its carpooling routing algorithm enables messages to travel together. Moreover, without using GPS, SDS routing leads to comparable routing performance by utilizing only limited location information in mobile condition.

(4) Low latency. Latency is determined by the efficiency of routing and data processing. Zone-based routing in SDS utilizes zones rather than individual nodes as a routing step unit, thus accelerating message transmission. Further, it reduces congestion due to many messages as in Directed Diffusion. In addition, unlike GHT that searches each attribute in a query and then performs merging operation for the final results, SDS leads to low latency without the need of a merging operation.

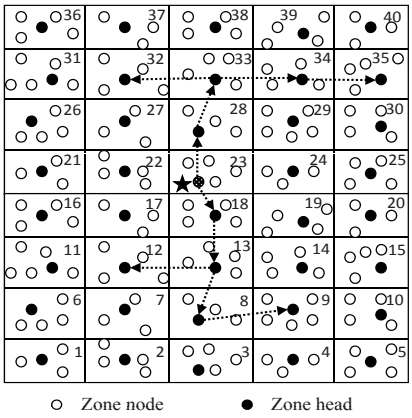


Fig. 1. SDS zones and data transmission.

3.2 Overlay Maintenance

Consider a large-scale WSN that is deployed in a vast field, in which sensors are disseminated randomly. Without the loss of generality, we assume that the field is a rectangle. SDS can also be extended to other shapes of fields. As shown in Figure 1, SDS divides the entire field horizontally and vertically to small rectangular zones. The division is based on the geographical layout of the entire area.

Before deployment, each node is configured to be aware of the zone layout by knowing: (1) The number of zones horizontally n_x and vertically n_y; (2) The zone ID assignment scheme, e.g., IDs are assigned to successive zones in a sequential order from left to right starting from the bottom-left zone; and (3) The ID and geographical location (x,y) of its own zone. Therefore, a node in the zone with ID_i can calculate the relative horizontal and vertical distances from any zone with ID_j by $\delta X_{i,j} = (ID_j - ID_i)\%n_x$ and $\delta Y_{i,j} = (ID_j - ID_i)/n_x$, respectively. The Euclidean distance between the two zones is $|ID_i, ID_j| = \sqrt{\delta X_{i,j}^2 + \delta Y_{i,j}^2}$. Additionally, the zone IDs of the up, down, left and right neighboring zones, if exist, can also be calculated by $ID_i + n_x$, $ID_i - n_x$, $ID_i - 1$, and $ID_i + 1$, respectively.

Each zone has a head which functions as the server for all other nodes (i.e., clients) in the zone. The head maintains the links to the neighboring clients within

its zone, and the heads in its neighboring zones. It periodically exchanges "hello" messages with its clients and neighboring zone heads. A head communicates with its neighboring heads by specifying the IDs of the neighboring zones in the messages.

In order to save energy, we assume every node has two interfaces to adapt to different transmission range: a short range interface and a long range interface. The short range interface enables nodes to communicate with other nearby nodes and is what is used most of the time. The long range interface enables adjacent head nodes to directly route messages when necessary.

Each sensor in the WSN has an identifier which is the consistent hash value [30] of its IP address. The identifiers of nodes in a zone are normalized to identifiers from 1 to N. To further reduce energy consumption, in each zone, all nodes except the head are in sleep mode. Nodes in sleep mode still can sense data, but rely on the head for other functions. The nodes rotate the responsibility of acting as the zone head in a round-robin manner in the order of their identifiers to balance the workload and energy consumption among nodes. Each node keeps a countdown timer. Once becoming a head, a node sets its countdown timer to the length of duty period L_h and triggers it to start the count down. With the knowledge of all the nodes in its zone, the head knows the next head among clients in the round-robin manner. On timeout, it transfers its duty to the next head and sends it the information of links of clients and neighboring heads.

The round-robin head election is suitable for the case when nodes have homogeneous capacity in terms of computing resource, energy and mobility. In the case when nodes have heterogeneous capability in computing and energy, nodes with larger capability have higher priority to be selected as heads. In a dynamic environment, nodes with lower mobility have higher priority to be selected as heads. The head's duty period can be adjusted to balance the head change overhead and the effect of load balance.

3.3 Data Processing and Mapping

After sensing an event, a sensor processes the event data, maps the data to a number of zones, and then stores the data to certain nodes in the zones. Each data item is described by a set of keywords. To process the sensed data, a sensor derives the keywords of the data through a strategy that allows the content of data to be described by keywords [31], [32], [33]. For example, the keywords *five car north* are derived from data *five cars are moving north*. We use d to denote a data item consisting of keywords v^d, represented by $d = (v_1^d, v_2^d, ..., v_m^d)$. A sensed data item is represented by a descriptor consisting of the following tuple: $< d, ID, t, s, (x, y) >$, where ID is the ID of the zone in which the data is sensed, t is the time when the data is sensed, s is the identifier of the sensor which sensed the data, and (x, y) is the exact geographical location where the event occurred.

Attribute	Keywords	Weight
Object	Car, Plane, Truck, etc.	0.3
Model	F-16, F-17, etc.	0.2
Color	Red, Purple, etc.	0.1
Direction	North, South, etc.	0.1
Division	AirForce, etc.	0.1
Pressure	*Integer*	0.1
Speed	*Float*	0.1
...	...	...

TABLE 1
Example of weight settings.

Note that a data keyword usually belongs to an event attribute such as object, direction, or speed. E.g, keyword "plane" belongs to the attribute "object". Table 1 illustrates an example of different attributes along with possible keywords. A WSN is usually for specific use (e.g., battlefield or habitat monitoring), so it is easy to retrieve the possible attributes for a specific application. Thus, we specify that SDS has a pre-defined attribute list geared towards its WSN application, and a detected event is described by the keywords belonging to the attributes. As shown in Table 1, SDS assigns weights to different attributes based on their importance in the event description. For example, "object" is a decisive factor to determine the similarity of two sensed events, so it is assigned a higher weight. Other attributes such as "speed", "direction" and "pressure" have less importance and hence are assigned lower weights.

Given m attributes in an attribute list associated with weight w_i $(1 \leq i \leq m)$ in a WSN application, a data item is denoted by $d = (v_1^d, v_2^d, ..., v_m^d)$, where v_i^d is the keyword for the i^{th} attribute associated with weight w_i. Then, the similarity of data item d_2 to data item d_1 is calculated by:

$$Similarity = \frac{\sum_{i=1}^{m} w_i \times B(v_i^{d_1}, v_i^{d_2})}{\sum_{i=1}^{m} w_i}, \qquad (1)$$

where $B(i, j)$ is a boolean value function, returning 1 when $v_i^{d_1} = v_i^{d_2}$ and 0 otherwise. For example, if a WSN application has an attribute list with the first five attributes in Table 1, the similarity of data item $Plane|F-16|Blue|Northwest|AirForce$ to $Plane|F-16|White|Southeast|AirForce$ is (0.3*1+0.2*1+0.1*0+0.1*0+0.1*1)/(0.3+0.2+0.1+0.1+0.1)=0.75.

For data mapping, SDS resorts to a locality-sensitive hashing function (LSH) [29] to transform d to a series of hash values. A detailed description of the LSH approach can be found in [29]. Data items having common keywords will have the same hash values, while similar data items will have similar hash values. The number of hash values of a data item can be flexibly set in LSH. Higher values tend to lead to fine-grained data clustering while lower values tend to lead to coarse-grained data clustering. Taking one resultant LSH hash value as an example, if the difference between d_1, d_2 and d_3 is $d_1 > d_2 > d_3$, their hash values conform to $h_{d_1} > h_{d_2} > h_{d_3}$, where h_d is the hash value of d.

In the mapping between data and zones, the ID differences between zones indicates the similarity between the data stored in the zones. To achieve this objective,

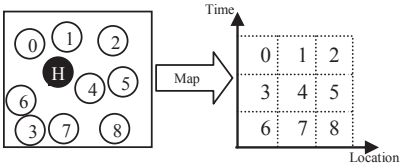


Fig. 2. Spatial-temporal data mapping.

a data item with hash value h is mapped to the first zone with $ID \geq h$. For example, a data item with $h = 5$ will be stored in zone 5. If all nodes in zone 5 fail, the data will be stored in zone 6. Thus, we can see that the distance between neighboring zones in one horizontal level indicates the similarity of data in the zones.

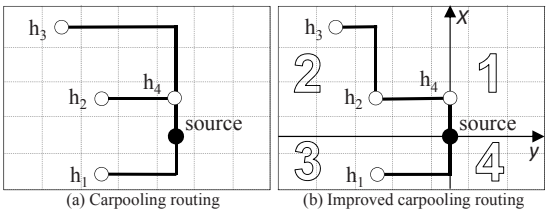
Within a zone, the data is further distributed among nodes according to their time and location. In static deployment, we first introduce a simple mapping. We then extend this mapping to dynamic network environment in Section 3.6. Figure 2 shows this data mapping within a zone in a spatial-temporal two-dimensional manner. Given a zone containing N client nodes, the zone head calculates $k = \lfloor \sqrt{N} \rfloor$. The identifiers of nodes in the zone are normalized to identifiers from 1 to N. The head virtually arranges the nodes into a $k \times k$ grid as shown in the figure. Assume there are S zones in the WSN in total. The head divides the range $[1, S]$ into k even parts. Taking a certain period of time, say one month, as a unit time interval T, the head divides T into k parts evenly. During T, for a node with identifier i, it is responsible for the data of events that occurred during the $(\lfloor i/k \rfloor + 1)^{th}$ time interval at the $(i\%k + 1)^{th}$ location.

For example, in Figure 2, there are 9 nodes with identifier 0-8 within a zone, then $\lfloor \sqrt{N} \rfloor = 3$. For node 4, $\lfloor 4/3 \rfloor + 1 = 2$, $4\%3 + 1 = 2$. Thus, it is responsible for the second $\frac{1}{3}$ of the total time interval during T, and the second $\frac{1}{3}$ of the total range of location. In other words, a data item with t and ID will be stored in the node located in line $t\%\frac{T}{k}$ and column $ID\%\frac{S}{k}$ in the two-dimensional space. Therefore, the data mapped to a zone is distributed among the zone nodes in balance.

3.4 SDS Routing Algorithm

In SDS, the operation of data storing and querying are performed by the SDS routing algorithm. We aim to develop a lightweight routing algorithm with high efficiency and low energy consumption. Also, in order to avoid generating high cost for centralized computing of routes in the requesters, we aim to provide a decentralized routing algorithm in which the requester or relay node selects the next relay node by itself. Moreover, a node selects the relay node without relying on geographical information, thus reducing energy consumption.

Recall that one data item has a series of hash values (say n values). Thus, one data item is stored in n nodes. When a node senses an event, it calculates n hash values of the data using LSH, $(h_1, h_2, h_3 \cdots, h_n)$. It then sends the event data $< d, ID, t, s, (x, y) >$ to its zone head along with the n destinations. Therefore, the head then sends n copies of data to destination zones with



(a) Carpooling routing (b) Improved carpooling routing

Fig. 3. Examples of two routing algorithms.

IDs equal to $h_i (1 \leq i \leq n)$ respectively along with other heads. After a data item is routed to the head of its destination zone, the head assigns the data to the node responsible for it. Using the data assignment method described in Section 3.3, the head identifies the destination node of the data based on the zone ID of the event collector and the moment that the event happened. It then forwards the data to the node. We propose the normal and improved carpooling routing algorithms for data storage and search.

3.4.1 Carpooling Routing Algorithm

In the carpooling routing algorithm, the source head determines the next hop among its neighbor heads based on each hash value. Assume the IDs of its neighboring zones are $ID_j (1 \leq j \leq 4)$. The next hop for each hash value h_i is the head in the zone with $min|ID_j, h_i| (1 \leq j \leq 4)$. That is, the next hop is the neighboring head that is the closest to the destination zone. Data copies targeting different destination zones but the same direction are very likely to have the same next hop. In order to save overhead, the source head conducts the carpooling operation. That is, rather than sending n pieces of data to n nodes directly, the source head only sends one copy to each different next hop indicating destination zones. After the next hop receives the data, it chooses the next hop in the same manner as the original head did. This process is repeated until the data is forwarded to its destination zones. Figure 1 presents an example of the routing algorithm.

By combining messages in the same direction together in routing, the carpooling algorithm decreases routing cost. However, it does not reduce the cost to its minimum. For example, as shown in Figure 3(a), the source head combines the messages to destinations h_2, h_3 and h_4 together, and forwards the message to h_4. The zone head of h_4 sends a message to h_2 and h_3, respectively. However, as shown in Figure 3(b), if the head sends one message to h_2 which further sends a message to h_3, it will further reduce the routing cost. Therefore, we propose the improved carpooling algorithm.

3.4.2 Improved Carpooling Routing algorithm

The goal of the improved carpooling routing algorithm is to combine the messages towards the same direction as much as possible and forward the combined messages as far as possible. When a source head originates a message with destinations $h_1, h_2, .., h_n$, it first divides the entire WSN area into four quadrants, i.e., Q_i $(1 \leq i \leq 4)$, using

the X and Y axes as shown in Figure 3(2). As introduced in Section 3.2, given the ID of a destination zone, a node can calculate the position of the destination zone relative to its own zone in the WSN layout. Thus, the source can locate the n destination zones with ID h_i $(1 \leq i \leq n)$ in the four quadrants, and get four subsets of destinations. A node on the X or Y axes can be divided into either the clockwise or counterclockwise quadrant. We classify such a node to the quadrant with more destinations in order to increase the possibility that it is in the same quadrant with more nodes in the routing, so that its message can be "carpoolled" with others. For example, in Figure 3, h_4 is classified into h_2.

For the subset of each quadrant $Q_i(1 \leq i \leq 4)$ with destinations $D^{Q_i} = h_1^{Q_i}, h_2^{Q_i}, ..., h_p^{Q_i}$, the source head s first identifies the *temporary* destination in Q_i, denoted by $h_t^{Q_i}$, and $|h_t^{Q_i}, s| = min|h_k^{Q_i}, ID_s|(1 \leq k \leq p)$. The list of the temporary destinations along with their destinations is $tempDests = \{(h_t^{Q_1}, D^{Q_1}); \cdots; (h_t^{Q_4}, D^{Q_4})\}$. The head s can then use the carpooling routing algorithm to send a message to each temporary destination along with the message's destinations.

Note that messages in the same direction are more likely to have the same next hop. In order to optimally combine the messages for different destinations with the same next hop, s calculates the next hops for the destinations in the vertical and horizontal directions, respectively, and chooses the list with less distinct next hops. Then, s sends one message to the next hop in the chosen group along with temporary destinations and their associated destinations. Upon receiving a message, a node conducts the same operation as s. Eventually, this recursive process leads the message to each destination at minimal cost. Algorithm 1 shows the pseudo-code for the improved carpooling algorithm.

3.4.3 Range-based Similarity Search

When a node searches for similar data, it can specify similarity degree. Recall that the difference of zone IDs represents the similarity degree of data in the zones. Thus, if a requester wants to receive more data with less similarity degree, it can specify a range r to indicate the zone range needed to be searched during data retrieval. For instance, for a hash value h, the zones with $ID \in [h - r, h + r]$ are searched. In routing, the head in the destination zone h forwards the query to the heads in zones $h - 1$ and $h + 1$. These heads further forward the query to zone $h - 2$ and $h + 2$, and so on until the heads in zone $h - r$ and $h + r$ receive the query. After a node receives a query, it checks whether the data it stores meets the similarity requirements using Equation (1) and responds with the desired data. For instance, if a query has:

$$\langle h_1, h_2, ..., h_n \rangle, similarity = 50\%, r = 1, \qquad (2)$$

then zones with IDs $\in [h_i - 1, h_i + 1](1 \leq i \leq n)$ will be searched, and data with similarity no less than 50% to the query will be retrieved.

```
1: Route_Msg(msg, dest[n])
2: /*send the msg to the nodes inside my zone*/
3: if dest.contains(my_zone()) then
4:     send_msg_to_node();
5:     dest.remove(my_zone());
6: end if
7: /*divide destinations to 4 quadrant sets*/
8: for each h in dest do
9:     Integer QID = calQuadrant(h)
10:    D[QID].add(h)
11: end for
12: /*calculate the temporary destination for each quadrant set*/
13: for i=1 to 4 do
14:    tempD = getClosestD(D[i], my_zone())
15:    /*generate next hops for the temporary destination in both
       horizontal and vertical directions*/
16:    next_hopH=closestNeighborH(my_zone(),tempD)
17:    next_hopV=closestNeighborV(my_zone(),tempD)
18:    next_hopsH.add(next_hopH, (tempD, D[i]))
19:    next_hopsV.add(next_hopV, (tempD, D[i]))
20: end for
21: next_hops=shortL(Carpool(next_hopsV),Carpool(next_hopsH))
22: /*send the msg to the next hop with a temporary destination and a
    destination list*/
23: for all key next_hop in next_hops do
24:    send_msg(next_hop, msg);
25: end for
```

Algorithm 1: Pseudo-code of the SDS improved carpooling routing conducted by a node.

3.5 Load Balancing

3.5.1 Storage Load Balancing

Storage load balancing is used to prevent imposing too high of a storage load on some nodes and better utilize the nodes with light storage load. A node's storage usage status is represented by the percentage of its used storage, denoted by S. The storage usage status of a zone with N nodes is calculated by $\sum_{i=1}^{N} S_i/N$. We define ϕ as a threshold of the percentage of a zone's used storage to indicate when a zone is at risk of being overloaded. Periodically, clients report their storage usage percentages to their head, and neighboring heads exchange zone storage usage.

When receiving a data storage request, a zone head first examines if its zone's storage usage has reached threshold ϕ. If so, the head node resorts to the neighboring zones. It forwards the request to the head of its most lightly loaded neighboring zones, and creates an index in itself indicating the host zone of the data. Later on, when the head receives a query for the data, according to the index, it forwards the query to the actual host neighboring zone of the data.

3.5.2 Routing Load Balancing

Routing load balancing is used to evenly utilize all zones to forward the query and storage requests. As shown in Figure 4, in the SDS routing scheme, the calculated shortest route between a source or relay and destination is not unique. We are interested in finding out the maximum number of possible

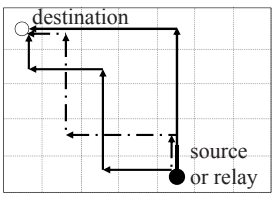


Fig. 4. Multiple shortest routes.

shortest routes between a pair of zone heads H_a with (X_a, Y_a) and H_b with (X_b, Y_b). To simplify the analysis, we assume $|X_a - X_b| = |Y_a - Y_b|$ and define $\mathcal{C} = |X_a - X_b|$. $\mathcal{C}$ is the maximum number of inflexion points in the horizontal and vertical routing process, respectively. Different shortest pathes have different number of inflexion points. The problem of choosing j inflexion points among $\mathcal{C}$ inflexion points is equivalent to the problem of putting $\mathcal{C}$ balls into j boxes, and the number of choices is $\binom{\mathcal{C}-1}{\mathcal{C}-k}$ [34]. For each choice with j inflexion points, there are 2^j possible routes. Consequently, the maximum number of possible shortest routes between H_a and H_b equals $\sum_{j=1}^{\mathcal{C}} \binom{\mathcal{C}-1}{\mathcal{C}-j} 2^j$. SDS distributes the routing load to all possible routes of each query by letting each query issuer list all the shortest calculated paths and randomly choose one at each time. Therefore, when two nodes frequently communicate with each other, the routing load will be evenly distributed among the nodes between them.

3.6 Dynamic Data Management

3.6.1 Tree-based Data Storage

In a mobile WSN, sensor nodes move around in the area. Node mobility has posed a challenge to the network management and data storage [35], [36]. SDS is also designed to operate in a dynamic environment where sensor nodes are mobile and may fail constantly.

As aforementioned, nodes with lower mobility have a higher priority of being selected as heads in a dynamic environment. According to the heads' signal strength, each client can locate its own head and know whether it has left its original zone. Also, according to the strength of the signal from its neighboring heads, a head can tell if it is moving out of its own zone. In order to enhance the localization accuracy, we can incorporate the state-of-the-art approaches [37], [38], [39] by utilizing beacons to enable a node detecting whether it has departed its original zone by examining the signal strength from the beacons (i.e., a set of pre-selected landmarks nodes). When a node departs its original zone and moves into a new zone, it informs the heads in both zones. Therefore, a zone head always knows the number of nodes in its zone. Before a head moves out of its original zone, it chooses the next node in a round-robin manner as a new head and becomes a client in the new zone. Note that the zones under node mobility are no longer strict rectangles, but the membership of each node is still strictly ensured to be unique.

The two-dimensional spatial-temporal storage space method described in Section 3.3 divides a zone into grids for data distribution among zone nodes. This method arranges each grid to have one node, but this is not sufficiently robust in a dynamic environment. Firstly, in a dynamic environment, the number of nodes is constantly changing. Thus, there may not be a node for a grid and some extra nodes may not be fully utilized to store data, especially when we seek to find a reasonable granularity. For example, if we want a 5×5 division for a zone

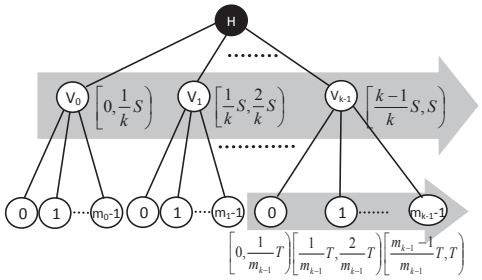


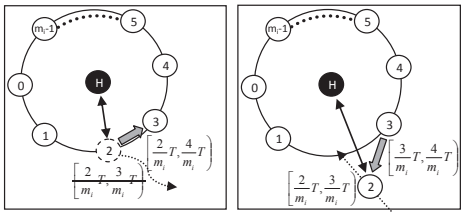
Fig. 5. Tree-based robust data storage structure.

originally containing 25 nodes, but has only 23 nodes due to node mobility, it will be costly to alter the storage structure and reassign duties to nodes. Secondly, the maintenance of the two-dimensional spatial-temporal storage space is difficult due to node mobility.

Therefore, we extend the two-dimensional storage space to an easy-to-maintain tree structure that is efficient under both static and dynamic environments. Figure 5 shows the tree structure. It is a generalization of the two-dimensional storage space in Figure 2. The tree root is the head of the zone that is responsible for maintaining the tree structure. The second level consists of k virtual nodes with identifier V_i $(i \in \{0, k\})$ having m_i leaves. Virtual nodes are responsible for non-overlapping spatial slices of the entire space interval. Therefore, every virtual node V_i is responsible for a spatial slice $[\frac{i}{k}S, \frac{i+1}{k}S)$, as shown in Figure 5. The virtual nodes are not real nodes and function to help keep the tree structure. The use of virtual nodes does not bring about extra communication cost, since they do not change the manner in which the head node communicates with leaf nodes.

Further, the leaves in the tree are real nodes that store data. Every virtual node V_i has m_i leaves, and the leaves are ordered from 0 to m_i. We call the order number as *leafId*. Each leaf node with leafId j $(0 \leq j \leq m_i)$ stores the data of time slice $[\frac{j}{k}T, \frac{j+1}{k}T)$. The number of leaves managed by a second-level virtual node is not fixed in order to adapt to different numbers of nodes and network dynamics. Recall that nodes within a zone have a consecutive sequence of normalized identifiers as shown in Figure 2. A leaf node j under virtual node i has the normalized identifier: $j + \sum_{x=0}^{i-1} m_x$.

To achieve a balanced initial state, when nodes are deployed, the number of virtual nodes k is set to $\lfloor \sqrt{N} \rfloor$, and the number of nodes under each virtual node is set to $\lfloor \frac{N}{k} \rfloor$. The tree-based structure is then maintained under node joins and departures. For robust maintenance of the structure, we constitute all nodes under one virtual node into a ring structure by adding a link between the first node and the last node. Figure 6(a) shows the logical ring structure of leaf nodes in one virtual node. Each node has a predecessor and successor. We define the first node whose leafId is equal to or follows leafId in the ring the successor of the leafId. A data item is assigned to a node with leafId j according to its time slice. If the node is not present, the data is stored in the successor of the leafId.



(a) Node departure (b) Node join

Fig. 6. Structure maintenance for node dynamism.

3.6.2 Node Departure and Failure

Before a client leaves, it notifies its zone head and its predecessor and successor. It then transfers its data to its successor. Its predecessor and successor then connect to each other. In Figure 6(a), node 2 originally stores the data in time slice $[\frac{2}{m_i}T, \frac{3}{m_i}T)$. Before it leaves, it transfers all of its data to node 3. As a result, node 3 is responsible for $[\frac{2}{m_i}T, \frac{4}{m_i}T)$.

Recall that nodes in a zone become the zone head in a round-robin manner. Before a head leaves its zone, it notifies the node that will become the next head. It then provides the new head with all the information stored in itself. The new head node conducts the operations for client departure in order to leave the ring, i.e., the control of its original virtual node. We will address the extreme case that all nodes in a virtual node have left in Section 3.6.4. Algorithm 2 describes the pseudo-code for the process of node departure.

When a node fails, it does not have the chance to inform its head, predecessor and successor. When a head node does not receive the signal from one of its client for a certain time period, it assumes the client fails. It then notifies the failed node's predecessor and successor to connect to each other. In a node failure, data in a failed node is lost. To handle the data loss, each zone head can have a copy of all data in its zone. After restarting, a failed node can ask for its data from the zone head.

```
1: Head_Leave()
2: /*choose a new head*/
3: new_head = get_nexthead()
4: transfer_info(new_head)
5: Client_Leave()
6: /*new head notifies its clients in the zone*/
7: new_head.build_connections()
8:
9: Client_Leave()
10: notify zone head
11: notify predecessor and successor
12: if have_data() then
13:     /*transfer data to its successor*/
14:     transfer_data(successor)
15: end if
16: return me
```

Algorithm 2: Pseudo-code for node departure.

3.6.3 Node Join

When a node joins in a new zone, it will firstly inform its new zone head, who has the information of the tree structure. It assigns the newly-joined node to a virtual node with the objective to balance the load among virtual nodes and the load among real nodes (i.e., leaves) under the same virtual node. Here, the load of a leaf node is the ring space owned by itself and the load of a virtual node is the number of leaves owned by itself. Recall that at the initial stage, all virtual nodes have the same number of leaves. Due to node mobility and failure, some virtual nodes will have more leaves while others have less. The head tries to add newly-joined nodes to less loaded virtual nodes. Specifically, it firstly finds a virtual node that has the least leaves. It then finds a client node that has more workload than any of the other nodes. Finally, the head inserts the new node into a position on the ring that releases the workload of the more loaded node. As shown in Figure 6(b), node 3 was responsible for $[\frac{2}{m_i}T, \frac{4}{m_i}T)$. After a new node joins in the system, the head positions it between node 1 and 3 by assigning it leafId 2. Then, $[\frac{2}{m_i}T, \frac{4}{m_i}T)$ is split into $[\frac{2}{m_i}T, \frac{3}{m_i}T)$ and $[\frac{3}{m_i}T, \frac{4}{m_i}T)$. Algorithm 3 shows the pseudo-code of node the join operation conducted by a zone head. $min\_load\_vnode()$ is to find a virtual node that has the least number of leaf nodes.

```
1: Client_Join(new_node, tree)
2: /*find a virtual node with the least number of leaves*/
3: vnode = min_load_vnode(tree)
4: insert_point = null
5: max_load = 0
6: for all node in vnode do
7:     /*find a virtual node that has maximum duty*/
8:     if node.check_load() > max_load then
9:         insert_point = node
10:        max_load = node.get_load()
11:    end if
12: end for
13: new_node.id = insert_point.id − 1
14: transfer_data(new_node, insert_point)
```

Algorithm 3: Pseudo-code of node join operation conducted by a zone head.

3.6.4 Load Balance

In order to achieve an appropriate granularity of temporal or spatial data storage, the load between virtual nodes should also be balanced. Load balancing between virtual nodes can be treated as a series of node departure and join operations. SDS sets a pre-defined threshold t_0 for the load difference between the virtual node with the maximum load, denoted by v_h, and the virtual node with the minimum load, denoted by v_l. The head periodically calculates the load difference between v_h and v_l, denoted by d. In the case that $d > t_0$, $d/2$ leaf nodes depart v_h and join in v_l. This process is repeated until t_0 is reached. Algorithm 4 shows the pseudo-code of the load balancing between virtual nodes.

4 Analysis of SDS

4.1 Analysis of the SDS Routing Algorithm

First, we analyze the routing cost of SDS with and without the carpooling algorithm. In SDS without the algorithm, a message is transferred to each of the n destinations separately. As we indicated earlier, message can be routed between neighboring zones when necessary.

```
1: Balance_Load(tree, t_0)
2: while true do
3:    v_h = max_load_vnode(tree)
4:    v_l = min_load_vnode(tree)
5:    /*find the difference between v_h and v_l*/
6:    d = v_h.load − v_l.load
7:    if d < t_0 then
8:       return
9:    else
10:      /*d/2 nodes leave the v_h, and join in v_l*/
11:      for i = d/2; i > 0; i − − do
12:         notify v_h.min_load_node() to leave v_h and join in v_l
13:      end for
14:   end if
15: end while
```

Algorithm 4: Pseudo-code of load balancing between virtual nodes.

Without loss of generality, in the analysis, we assume a message is routed horizontally first and then vertically. Let c be the cost of one data transmission between a pair of neighboring head nodes. We use (X, Y) to represent a zone, where X and Y denote the sequence number of the zone in the horizontal and vertical directions. For a single query issued by node in zone (X_0, Y_0), to nodes in zone $(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$, SDS without carpooling routing algorithm sends out n requests to n different zones. The cost of the query routing is:

$$C^{w/o}_{X_0,Y_0,n} = \sum_{i=0}^{n} (|X_0 - X_i| + |Y_0 - Y_i|) \times c \tag{3}$$

The carpooling algorithm reduces cost by combining the paths of requests with a common path. We estimate the upper bound and lower bound of this carpooling routing cost denoted by C^w. It can be seen that the longest distance the data traversed towards right and left directions are

$$max\{X_0 - X_1, ..., X_0 - X_n\}, \tag{4}$$

and

$$|min\{X_0 - X_1, ..., X_0 - X_n\}|. \tag{5}$$

Since the routing is firstly on the horizontal direction, the horizontal routing cost can be at most minimized to $(max\{X_0 - X_1, ..., X_0 - X_n\} + |min\{X_0 - X_1, ..., X_0 - X_n\}|$. Considering the normal vertical routing cost, we have

$$\begin{aligned} C^w_{X_0,Y_0,n} \leq (&max\{X_0 - X_1, ..., X_0 - X_n\} + \\ &|min\{X_0 - X_1, ..., X_0 - X_n\}| + \\ &\sum_{i=0}^{n} |Y_0 - Y_i|) \times c. \end{aligned} \tag{6}$$

In the ideal situation where carpooling can save the horizontal routing cost as well as the vertical routing cost, we have the lower-bound of the querying cost:

$$\begin{aligned} C^w_{X_0,Y_0,n} \geq (&max\{X_0 - X_1, ..., X_0 - X_n\} + \\ &|min\{X_0 - X_1, ..., X_0 - X_n\}| + \\ &max\{Y_0 - Y_1, ..., Y_0 - Y_n\} + \\ &|min\{Y_0 - Y_1, ..., Y_0 - Y_n\}|) \times c. \end{aligned} \tag{7}$$

Given Equation (3) and (6), it can be observed that

$$\begin{aligned} C^{w/o}_{X_0,Y_0,n} - C^w_{X_0,Y_0,n} \geq (&\sum_{i=0}^{n} |X_0 - X_i|) - \\ &(max\{X_0 - X_1, ..., X_0 - X_n\} + \\ &|min\{X_0 - X_1, ..., X_0 - X_n\}|) \times c > 0. \end{aligned} \tag{8}$$

Equation (8) shows that even in the worst case, SDS with carpooling algorithm, $C^w_{X_0,Y_0,n}$, is less than SDS without the algorithm, $C^{w/o}_{X_0,Y_0,n}$. In addition, the cost saving increases as n grows.

Because the requester and queried nodes are all randomly distributed, we assume all zones have the same probability to issue queries. With the assumption that there is only one node in each zone that issues a query, then the expected cost of a query with carpooling mechanism is:

$$E_{C^w} = \frac{\sum_{Y=0}^{\sqrt{Z}} \sum_{X=0}^{\sqrt{Z}} C^w_{X,Y,n}}{Z}, \tag{9}$$

where Z is the number of zones in the network.

4.2 Analysis of the Spatial-temporal Storage and Querying

We will first analyze the storage cost of SDS. We use S^{st}_c to denote the spatial-temporal pattern storage cost in SDS, and S_c to denote the storage cost without the SDS's spatial-temporal storage scheme. Without the spatial-temporal storage scheme, all data matched to one zone is stored in each node in the zone, while this storage scheme distributes the data among the zone nodes according to spatial and temporal order. We use β to denote the resultant fraction of all data stored in one node due to the distribution. Suppose that every zone contains N nodes on average, we have

$$\frac{S_c}{S^{st}_c} = \frac{s \times N}{(s \times \beta) \times N}, \tag{10}$$

where s is the size of the memory used for data storage in a node. We assume that data is evenly distributed in both spatial and temporal space. Thus, β equals to $\frac{1}{N}$. Therefore,

$$\frac{S_c}{S^{st}_c} = N. \tag{11}$$

The result means that spatial-temporal data storage scheme in SDS reduces the memory consumption.

Now, we analyze the querying cost of SDS based on spatial-temporal data storage compared with GHT [15] for large-range queries. For a multi-attribute (i.e., keyword) query, GHT hashes each keyword for searching and merges the located data as the final results. For a spatial-temporal range query, GHT breaks the range query to separate sub-queries. For example, for query "what are the airplanes located in Chicago between 1pm to 4pm?", GHT breaks the time range to time points 1pm, 2pm, 3pm, 4pm or smaller grains. For a GHT query, assuming the number of sub-queries of each attribute has an expected value $\bar{a}$, then the total routing cost for one query is

$$C^{st}_{GHT} = E_{GHT} \times A \times \bar{a}, \tag{12}$$

where E_{GHT} is the expected cost of one source-destination transmission in GHT and A is the number of attributes in the query. According to SDS data

distribution, nodes within a zone are arranged in both spatial and temporal order, thus data within a range can be retrieved by one query. Recall that SDS sends n requests to n zones for each query, and E_{C^w} is the expected routing cost for a inter-zone data transmission. We use c_z to denote the routing cost for a intra-zone data transmission. Then, the total routing cost is

$$C^{st}_{SDS} = n \times (c_z + E_{C^w}) = n \times (c_z + E_{SDS} \times n), \quad (13)$$

where E_{SDS} denotes the SDS's expected routing cost to only one zone. By comparing C^{st}_{GHT} and C^{st}_{SDS}, we have

$$\frac{C^{st}_{GHT}}{C^{st}_{SDS}} = \frac{E_{GHT} \times A \times \bar{a}}{n \times (c_z + E_{SDS} \times n)} = \frac{E_{GHT}}{E_{SDS}} \times \frac{A}{n} \times \frac{\bar{a}}{n} \times \frac{n}{\frac{c_z}{E_{SDS}} + n}. \quad (14)$$

Recall that SDS routes a message horizontally first and then vertically. GHT can conduct diagonal routing. Therefore, GHT can at most reduce the SDS routing cost by $\sqrt{2}$ times for one source-destination routing. Hence,

$$E_{GHT} \leq E_{SDS} \leq \sqrt{2} E_{GHT}. \quad (15)$$

Sensors often generate data with many attributes to meet analysis need and n is usually a small number, in most cases,

$$A > n. \quad (16)$$

For a query with a large spatial-temporal range,

$$\bar{a} \geq n. \quad (17)$$

Because the cost of inter-zone transmission (i.e., E_{SDS}) usually consumes more energy than intra-zone transmission (i.e., c_z), we have

$$\frac{n}{\frac{c_z}{E_{SDS}} + n} \approx 1. \quad (18)$$

By applying Equations (15)(16)(17)(19) to Equations 14, we have

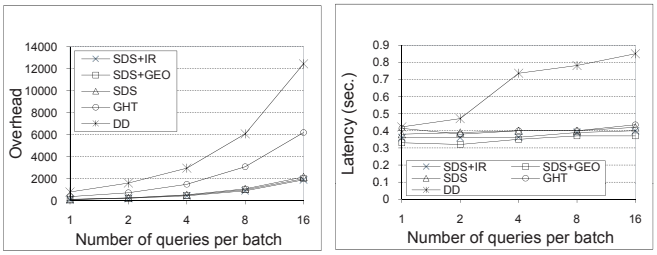
$$\frac{C^{st}_{GHT}}{C^{st}_{SDS}} > 1. \quad (19)$$

Therefore, we can conclude that in a scenario where data has a series of attributes or a large spatial or temporal querying range, SDS achieves better performance over GHT in spatial-temporal search in terms of routing cost. This conclusion is confirmed by the experiment results in Section 5.1.

5 Performance Evaluation

We used *The One* simulator [40] to evaluate the performance of SDS with the improved carpooling routing algorithm (SDS+IR) and SDS with the normal carpooling routing algorithm (SDS) in comparison with Directed Diffusion (DD) [9] and GHT [15]. For a multi-attribute (i.e., keyword) query, DD uses broadcasting to search all desired data. GHT hashes each keyword for searching and merges the located data as the final results. For reference, we also included the results of SDS using geographical routing [15] rather than the ID-based carpooling routing algorithm, denoted by SDS+GEO. It routes data directly from a requester to destination nodes.

The test scenario is a 400m×400m rectangular field, in which 400 nodes are randomly and independently disseminated. The field is divided into 16 data zones,



(a) Overhead (b) Latency

Fig. 7. Performance of spatial-temporal data search.

each of which has an area of $25m^2$ containing 25 nodes. The nodes located in the boundary zones of the WSN area send event data at a speed of 1kbps for 1400s. Each event data item has a randomly chosen size in [10,100] bytes, and contains time and location along with ten different keywords. The number of hash values for a data item after the LSH operation was set to 5. Before data querying operation, we gave each test 80s to distribute event data. The ways how nodes generate queries are different in different experiments and will be explained later. We used the following metrics to test the performance of different methods.

(1) Overhead. This is the sum of the products of path lengths and message weights, which is the length of a message divided by the average message length. We use message weight rather than length in order to conduct fair comparison since the message lengths of queried results of different approaches are different. This metric measures the cost of data querying and reflects energy consumption cost.

(2) Latency. This is the time period between a query is issued and a response is received, which reflects the effectiveness of a method in quick data retrieval.

(3) Total number of hop counts. This is the number of hops in a query routing, and it reflects the efficiency of the routing algorithms.

(4) Discovery rate. This is defined as the number of retrieved similar data items divided by all the existing similar data items. This metric reflects the effectiveness of a data storage method in similarity data searching.

(5) Success rate. This is defined as the number of successfully resolved queries divided by the total number of queries. This metric reflects the effectiveness of a data storage method in a dynamic WSN environment.

(6) Total number of discovered events. This is the number of discovered events using a search method. Comparing it with the existing events matching a query, the effectiveness of similarity search can be measured.

(7) Overlay maintenance cost. This is the average number of communication messages sent by a node in a zone for overlay maintenance, and it reflects the energy consumption cost for overlay maintenance.

5.1 Spatial-Temporal and Range Querying

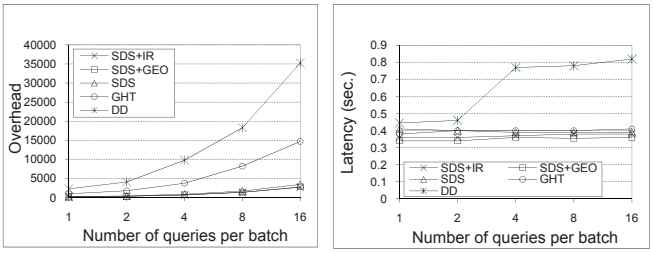
In this experiment, nodes generate queries targeted at events occurring within a time interval of 2s and a location range of 20. We define a *batch* as a group of

queries, in which the number of queries was set to $2^i (0 \leq i \leq 4)$. To send a batch of a queries, we randomly selected a nodes to generate queries with an interval of 0.1s between selections. After all queries in the batch were resolved, another batch was sent out. We generated 400 queries in total and calculated the average overhead per batch and latency per query as the experimental results. This setting applies to Figure 7, 8, 9.

Figure 7(a) shows the overhead of different methods. We can observe that SDS+IR, SDS and SDS+GEO generate the least overhead, DD generates the most overhead, and GHT falls in the middle. DD uses broadcasting for data querying, leading to significantly more messages, hence more overhead. For a query with ten keywords, a requester in GHT sends out one query for each keyword. Therefore, in GHT, much more data is routed back to the requester. In addition, GHT routing utilizes node as a step unit rather than a zone. As a result, it leads to much higher overhead than those of the SDS-based schemes. In contrast, no matter how many keywords a query has, the SDS-based schemes always sends five queries and it only returns the desired data. Moreover, the carpooling routing algorithm utilizes a zone as a step unit, leading to much less routing hops. We also observe that SDS and SDS+IR have comparable performance to SDS+GEO, which takes the geographically shortest path with the aid of GPS. This result implies a high efficiency of the SDS carpooling routing algorithms. The experimental results show that SDS+IR reduces around 15% overhead of SDS on average, indicating that SDS+IR is advantageous over SDS because of the improved carpooling. The improvement is not obvious because most of the overhead is contributed by the communication between head and clients in a zone as there are only 16 zones while 25 nodes in a zone.

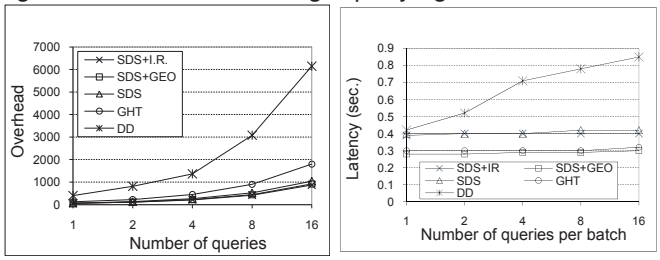
Figure 7(b) shows the average latency of each method. We can see that DD leads to a much higher latency than others. When the number of queries increases from 2 to 4, the latency of DD grows sharply due to its excessive messages and traffic congestion. SDS and GHT have almost the same latency, while SDS+IR and SDS+GEO produce the least latency. As previously mentioned, GHT returns much more data. Thus, the increased traffic causes congestion, resulting in a higher latency. ID-based routing in an SDS-based scheme without the aid of GPS may not always take the shortest path. However, SDS has less routing traffic, which reduces the possibility of congestion. With geographical routing, SDS+GEO leads to a lower latency due to less traffic and hops. SDS+IR produces latency comparable to SDS+GEO due to its improved carpooling, which further reduces the routing traffic and avoids traffic congestion in the heads.

Range querying is used to find data items within a certain range of similarity degree for a query. The range was set to 3 in this experiment. Since GHT is not locality-preserving in data storage, its exact-mapping querying cannot locate similar data. For comparison, we still include its querying results for the range $[h-r, h+r]$.



(a) Overhead (b) Latency

Fig. 8. Performance of range querying.



(a) Overhead (b) Latency

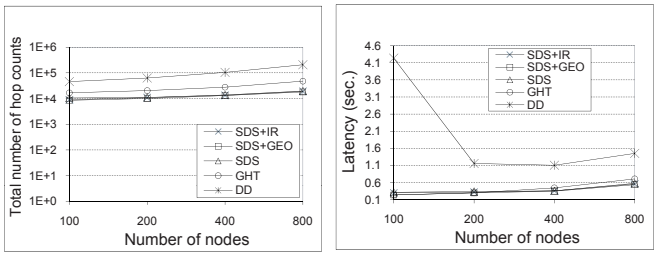
Fig. 9. Performance of single-result data querying.

GHT then needs to query each of the IDs in this range. Figure 8(a) plots the overhead of the methods. We observe that DD produces dramatically higher overhead than others due to its broadcasting. Also, we find that GHT's overhead increases greatly as the number of queries increases. This is because GHT sends out six more queries then the SDS-based schemes for every attribute corresponding to the range size. SDS+IR, SDS and SDS+GEO still generates the least amount of overhead. In range querying, the destination zones only need to forward the queries to their neighboring zones, which generates slightly more overhead. In addition, they utilize a zone as a step unit, leading to much less routing hops compared with GHT, which uses a node as step unit. The results show that the range querying provided by SDS-based methods are the most efficient among the approaches. The experiment results show that on average, SDS+IR reduces the overhead of SDS+GHT and SDS by 5% and 23%, respectively. This confirms the advantage of the improved carpooling algorithm.

Figure 8(b) shows that DD's latency increases dramatically due to congestion as traffic grows. GHT and SDS-based schemes have similar latencies. GHT takes the shortest path, but its latency is affected by the congestion caused by the increased traffic. SDS-based schemes do not need to send as many queries as GHT since they rely on neighbors to forward queries, thus reducing traffic and congestion. SDS+GEO has the least latency as a combined effect of the less traffic of SDS and short routing path of geographical routing. Also, SDS+IR achieves approximately the same latency as SDS+GEO because it further reduces the traffic and the congestion in heads with the improved carpooling routing.

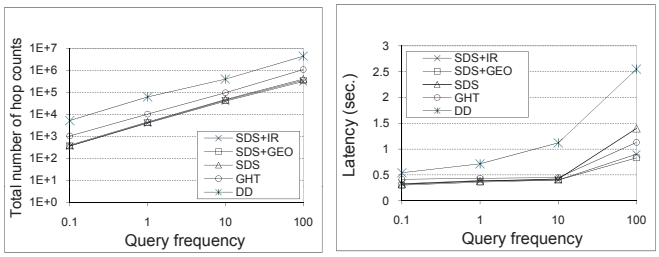
5.2 Single-Result Querying Performance

In order to compare the performance of each approach without the influence of congestion due to large amount



(a) Overhead (b) Latency

Fig. 10. Performance of scalability.



(a) Overhead (b) Latency

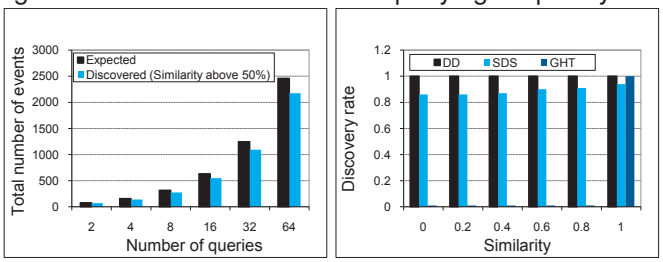
Fig. 11. Performance of different querying frequency.

of returned data, we tested the performance for queries with only one returned data item. The queries do not have a time or location specification. Figure 9(a) shows the overhead. We can see that SDS-based schemes produce the least overhead, while DD produces the most. The overhead of DD grows sharply as the number of queries increases due to its broadcasting. We also note that GHT generates more overhead than the SDS-based schemes. The SDS-based schemes send a query to five zones as a result of LSH operation. GHT sends a query to ten destination nodes and forwards the query along nodes rather than zones, thus resulting in a higher overhead. The experiment results show that SDS+IR averagely reduces the overhead of SDS and SDS+GEO by 23% and 12%, respectively, because SDS+IR can further reduce the number of hops with carpooling.

Figure 9(b) shows the latency of each method. We observe that DD leads to the highest latency and GHT produces less latency than the SDS-based schemes, but performs only slightly worse than SDS+GEO. As the number of queries increases, DD generates more messages, which leads to congestion and longer latency. Without GPS, SDS uses an ID-based routing algorithm that may not take the shortest path. On the other hand, SDS+GEO routes query directly to a node along the best path. Therefore, SDS and SDS+IR have higher latency than SDS+GEO. SDS+IR has nearly the same performance as SDS. This is because in single-result querying, the returned data results are much less than range-based querying and do not incur much traffic congestion. Since SDS+GEO has less traffic than GHT due to less messages, it has the least latency.

5.3 Scalability

The setting of this experiment is the same as the previous experiments except that the total number of queries was set to 200. Figure 10(a) shows the total number of hops. It demonstrates that DD's total number of hops is much higher and grows faster than others, which shows that DD has poor scalability. On the contrary, the total number of hop counts of SDS-based schemes and GHT grows relatively slowly, which implies the high scalability of these approaches. We find that the number of hops of SDS-based schemes remain fairly stable. This implies that these schemes have relatively stable routing performance in different scale WSNs.



(a) Discovered events (b) Discovery rate

Fig. 12. Performance of similarity searching.

Figure 10(b) illustrates the latency of each approach. DD has a dramatically higher latency than other methods especially when there are 100 nodes. DD broadcasts 200 queries among the small number of nodes, which makes it more likely to generate congestion. In contrast, SDS+IR, SDS, GHT and SDS+GEO exhibit similar low latency performances across varying node numbers. Therefore, they possess a high scalability.

Next, we tested the performance of different approaches with different querying frequencies. We randomly chose nodes to send queries for 70s with the querying frequency varied from 0.1 to 100 queries/s. Every query returned one data item. Figure 11(a) shows that the number of hop counts for all approaches increases linearly as the querying frequency increases. DD performs the worst because its broadcasting leads to much more traffic. SDS-based schemes have less total hop counts than GHT due to the same reason as explained in Section 5.2. The figure shows that SDS+GEO, SDS+IR and SDS are almost identical, but SDS+GEO and SDS+IR actually require fewer hops. The reason is that SDS+GEO routes a query using geographic information whereas SDS depends on a logical ID-based routing. Furthermore, SDS+IR reduces hop counts of SDS with the improved carpooling routing.

In Figure 11(b), the latency of all approaches grows slightly when the querying frequency increases from 0.1 to 10 queries/s. It then begins to grow sharply when the frequency increases to 100 from 10 queries/s. In this case, GHT performs better than SDS. Since the number of zones is less than the number of nodes, under a heavy traffic load, routing relying on the zone heads causes them to become more easily congested than by simply relying on the nodes. SDS+GEO is the least sensitive to the increase of querying frequency. It has less traffic than GHT as described above and it uses greedy forwarding.

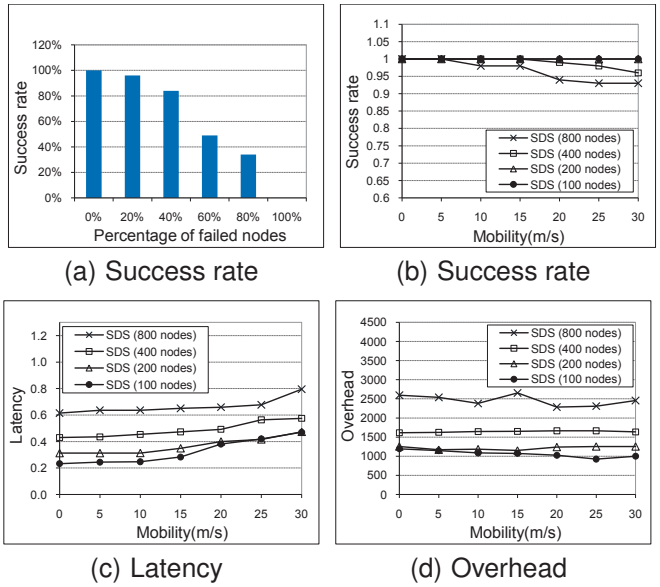


Fig. 13. Performance in a dynamic WSN with node failures and mobility.

Therefore, it benefits from less congestion. The latency produced by SDS+IR is less than SDS and very close to SDS+GEO under high querying frequencies. This is because its improved carpooling routing algorithm helps to reduce traffic congestion.

5.4 Similarity Searching

The setting of this experiment is the same as the previous experiments except the total number of queries was set to 100, and the number of queries per batch was set to $2^i(1 \leq i \leq 6)$. We set the query range of SDS to $r = 3$. All queries had specification of 50% similarity. Figure 12(a) shows the number of discovered data items with similarity no less than 50% in SDS and the number of actual such data items in the system. We find that SDS can always discover 90% of such data items. The results confirm that SDS is highly effective in similarity search.

Figure 12(b) shows the discovery rate of each approach in terms of the similarity between the discovered data and the query. Discovery rate is defined as the percent of data items that have a certain similarity to a query that can be discovered. GHT can only return data with 100% similarity due to its exact matching. Data in GHT is hashed to nodes using consistent hash functions. Thus, GHT fails to supply similarity searching. Unlike GHT and SDS, DD broadcasts queries to nodes and consequently achieves a 100% percent discovery rate for each similarity. However, its high discovery rate brings high overhead. SDS provides an optimized trade-off between overhead and discovery rate of similar data. The results show SDS obtains a discovery rate above 85% percent while avoiding the high overhead found with DD.

5.5 Performance in a Dynamic WSN

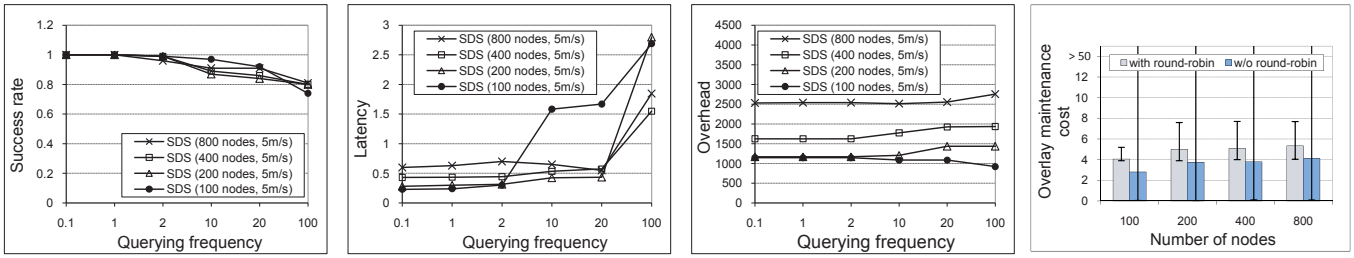
This experiment tests the performance of SDS with node failures and node mobility. Figure 13(a) presents the success rate with different percentage of failed sensor nodes. As expected, the success rate decreases as the percentage of failed nodes increase. This is because when a sensor node fails, event data may be lost on the way to its destination node where it is to be saved. In addition, all data in the failed nodes is also lost. Therefore, the responses of a query may not return all queried data.

This test was conducted in a WSN with $100 \times 2^i(0 \leq i \leq 3)$ nodes. Each zone has 25 nodes. The moving speed of the sensor nodes was set to 0-30m/s with 5 increase in each step. The queries were issued at a speed of 10 queries per second for a period of 40s. Figure 13(b) illustrates the success rate when the sensor nodes move at different speeds. It demonstrates that SDS achieves more than a 92% success rate when the nodes move at a speed of 30m/s. This shows the stability of SDS routing algorithm. Due to SDS's dynamic data management, a node can always obtain the event data in a system that it queries even under node mobility. We can also observe that faster mobility leads to slightly lower success rates due to the more frequent node joins and departures. Another phenomenon that can be observed is the increased success rate when there are fewer nodes. This is because the number of nodes per zone is the same in the different scale networks. Thus, a larger-scale network has more zones and, subsequently, generates longer message routing paths. Longer routing paths increase the probability of message transmission failure in a dynamic network.

Figure 13(c) shows the average latency of data transmission as a function of moving speed. Firstly, we can see that as the network scale exponentially grows, the transmission delay increases as well due to the increasing number of nodes that a message needs to traverse. For a network scale of 800, 400 and 200 nodes, the latency does not increase dramatically with higher moving speed, this demonstrates the high scalability of our dynamic management mechanism. Since the messages are generated and spread at the same speed in different scale networks, the network with 100 nodes suffers from more messages per node. This produces more congestion and leads to a higher latency escalating rate.

Figure 13(d) demonstrates the generated overhead versus node moving speed. It shows that this value increases as the network scale increases and does not change greatly as the node's moving speed increases. Networks with a size 100 and 200 generate nearly the same overhead at low moving speed. This is because when there are 100 nodes with the same amount of traffic, more message retransmissions are issued. These results show that SDS is suitable for a dynamic environment with mobile nodes.

Querying frequency influences the performance of WSNs. To evaluate the performance of SDS with different querying frequencies, we conducted our test with different querying frequencies and the node's moving speed of 5m/s. Figure 14(a) shows the success rate as a function of querying frequency. We see that as the fre-



(a) Success rate (b) Latency (c) Overhead

Fig. 14. Performance in a dynamic WSN with different querying frequency.

Fig. 15. The cost of overlay management.

quency increases, the proportion of delivered messages drops due to network congestion caused by an increased traffic load. Specifically, the network with only 100 nodes has the quickest falling success rate because it is the most heavily loaded. SDS can still keep its success rate about 80% in larger network scales.

In Figure 14(b), as the frequency of querying increases exponentially, the delay in the different scale networks rises. We see the increment in networks of more than 200 nodes is not apparent when querying frequency is between 0.1-10. This is because the amount of traffic does not yet cause significant congestion. We also see that every network suffers a sharp increase when the querying frequency increases to 100, as the messages are buffered and needs more time to transmit. The network with 100 nodes has an earlier increase in delay due to its scale and its processing of the same amount of messages as networks of other scales.

The overhead with different querying frequency is shown in Figure14(c). We observe that the overhead of all networks does not change much when the querying frequency is below 10. This shows SDS does not suffer from message retransmission. When the querying frequency becomes higher, the overhead of networks with more than 200 nodes goes up slightly, which indicates that a small amount of messages are retransmitted. A decrease in overhead happens on the network with 100 nodes under high querying frequency because the congestion is excessive and fewer messages have a chance to be sent.

5.6 Cost of Round Robin Zone Head Election

In order to show the cost of the round-robin zone head election, we measured the average, 1^{st} and 99^{th} overlay maintenance cost of SDS with and without the round-robin head election algorithm, as shown in Figure 15. We set the communication interval between a zone head and its clients to 60s and the interval between two elections of head nodes to 180s. We see that the SDS with round-robin generates a little higher average load than the SDS w/o round-robin. Also, the latter exhibits a much larger variance than the former. Thus, the cost caused by the round-robin head election is rewarded by the even distribution of the communication load among all of the zone nodes. Without the round-robin algorithm, the fixed head nodes are responsible for most of the communication cost and may be overloaded. Also, the low average cost values shows that the overhead in maintaining the zone-overlay is acceptable.

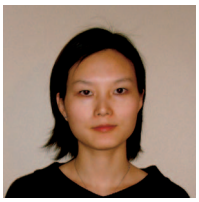
6 Conclusion

This paper proposes a distributed spatial-temporal similarity data storage scheme (SDS). Based on LSH, SDS efficiently disseminates data in a WSN such that similar event data is mapped to the same or nearby WSN neighborhood(s). This enables SDS to offer similarity searching service. SDS also provides spatial-temporal data searching by classifying data in a neighborhood into a two-dimensional or a tree data storage structure consisting of neighborhood nodes. Further, the SDS carpooling routing algorithm efficiently routes queries or data without relying on GPS. The experimental results show the distinguishing features of spatial-temporal similarity data searching of SDS. SDS not only shows superior performance over Directed Diffusion and GHT in terms of overhead and flexibility, but also exhibits comparable latency to GHT which employs geographical routing. SDS also achieves high reliability and stable performance in a dynamic environment.

References

[1] B. Krishnamachari, "networking wireless sensors," *Cambridge University Press, ISBN-10 0-521-83847-9*, 2005.
[2] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage," in *Proc. of WSNA*, 2002.
[3] G. Campobello, A. Leonardi, and S. Palazzo, "A novel reliable and energy-saving forwarding technique for wireless sensor networks," in *Proc. of MobiHoc*, 2009.
[4] G. Pottie, "Wireless Integrated Network Sensors," *Communications of the ACM*, no. 51–58, 2000.
[5] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Pro. of MMCN*, 2002.
[6] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, W. J. Kaiser, and H. O. Marcy, "Wireless integrated network sensors: low power systems on a chip," in *Proc. of ESSCIRC*, 1998.
[7] Y. Yao, X. Tang, and E. Lim, "In-network processing of nearest neighbor queries for wireless sensor networks," in *Proc. of DASFAA06*, 2006.
[8] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a Sensor Network Expedition," in *Proc. of EWSN*, 2004.
[9] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *Proc. of Mobicom*, 2000.
[10] W. Zhang, G. Cao, and T. L. Porta, "Data Dissemination with Ring-Based Index for Wireless Sensor Networks," in *Proc. of ICNP*, 2003, pp. 305–314.
[11] S. Madden, M. J. Franklin, and J. M. H. W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks," in *Proc. of OSDI*, 2002.

[12] F. Ye and G. Zhong, "GRAdient Broadcast: A Robust Data Delivery Protocol for Large Scale Sensor Networks," *WINET*, 2005.
[13] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang, "Ttdd: Two-tier data dissemination in large-scale sensor networks," *Wireless Networks*, vol. 11, pp. 161–175, 2002.
[14] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Grovindan, L. Yin, and F. Yu, "Data-centric storage in sensornet with ght: A geographic hash table," in *Proc. of MONET*, 2003.
[15] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, and L. Yin, "Data-centric storage in sensornets with GHT, a geographic hash table," *MONET*, vol. 8, pp. 427–442, 2003.
[16] X. Li, Y. J. Kim, and W. Hong, "Multi-dimensional range queries in sensor networks," in *Proc. of SenSys*, 2003.
[17] D. Ganesan, "DIMENSIONS: Why do we need a new data handling architecture for sensor networks," in *Proc. of the ACM HotNets*, 2002, pp. 143–148.
[18] D. Ganesan, A. Cerpa, Y. Yu, D. Estrin, W. Ye, and J. Zhao, "Networking issues in wireless sensor networks," *JPDC*, 2004.
[19] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker, "Difs: A distributed index for features in sensor networks," in *Proc. of SNPA*, 2003.
[20] J. Li, J. Jannotti, D. S. J. De, C. David, R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *Proc. of MobiCom*, 2000.
[21] J. Newsome and D. Song, "GEM: Graph EMbedding for routing and data-centric storage in sensor networks without geographic information," in *Proc. of SenSys*, 2003.
[22] A. Caruso, S. Chessa, S. De, and R. Urpi, "GPS free coordinate assignment and routing in wireless sensor networks," in *Proc. of IEEE INFOCOM*, 2005, pp. 150–160.
[23] P. Desnoyers, D. Ganesan, and P. Shenoy, "Tsar: A two tier sensor storage architecture using interval skip graphs," in *Proc. of SenSys05*. ACM Press, 2005, pp. 39–50.
[24] C. T. Ee and S. Ratnasamy, "Practical data-centric storage," in *Proc. of NSDI*, 2006.
[25] M. Aly, K. Pruhs, and P. K. Chrysanthis, "KDDCS: A load-balanced in-network data-centric storage scheme in sensor network," in *Proc. of CIKM*, 2006, pp. 317–326.
[26] F. Bian, X. Li, R. Govindan, and S. Schenker, "Using hierarchical location names for scalable routing and rendezvous in wireless sensor networks," in *Proc. of SenSys*, 2004, pp. 305–306.
[27] J. Xu, X. Tang, and W. chien Lee, "A new storage scheme for approximate location queries in object tracking sensor networks," *IEEE TPDS*, vol. 19, pp. 262–275, 2008.
[28] M. Li and Y. Liu, "Rendered path: range-free localization in anisotropic sensor networks with holes," in *Proc. of MobiCom*, 2007.
[29] H. Shen, T. Li, and T. Schweiger, "An Efficient Similarity Searching Scheme Based on Locality Sensitive Hashing," in *Proc. of ICDT*, 2008.
[30] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in *Proc. of STOC*, 1997, pp. 654–663.
[31] W. Nejdl, W. Siberski, M. Wolpers, and C. Schmnitz, "Routing and clustering in schema-based super peer networks," in *Proc. of IPTPS*, 2003.
[32] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu, "Data management for peer-to-peer computing: A vision," in *Proc. of WebDB*, 2002.
[33] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov, "Piazza: Data management infrastructure for semantic web applications," in *Proc. of WWW*, 2003.
[34] "In how many ways can m balls be distributed into n boxes?" http://www.fen.bilkent.edu.tr/ otekman/disc/usef.pdf.
[35] G. Wang, G. Cao, T. L. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *Proc. of IEEE INFOCOM*, 2005.
[36] L. Hu and D. Evans, "Localization for mobile sensor networks," in *Proc. of MobiCom*, 2004.
[37] F. Liu, X. Cheng, D. Hua, and D. Chen, "Location discovery for sensor networks with short range beacons," *IJAHUC*, 2009.
[38] R. Fonseca, S. Ratnasamy, J. Zhao, and C. T. Ee, "Beacon vector routing: scalable point-to-point routing in wireless sensornets," in *Proc. of NSDI*, 2005.
[39] N. Bulusu, J. Heidemann, and D. Estrin, "Gps-Less Low-Cost Outdoor Localization For Very Small Devices," *IEEE Personal Communications Magazine*, vol. 7, no. 5, pp. 28–34, 2000.
[40] "The one simulator. http://www.netlab.tkk.fi/."



Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the Department of Electrical and Computer Engineering, and the Director of the Pervasive Communications Laboratory of Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on peer-to-peer and content delivery networks, mobile computing, wireless sensor networks, and grid computing. Her research work has been published in top journals and conferences in these areas. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a member of the IEEE and ACM. She is Microsoft Research Faculty Fellow of 2010.

Lianyu Zhao received the BS and MS degrees in Computer Science from Jilin University, China. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include wireless sensor network, routing protocols, applications and security issues in P2P networks.

Ze Li received the BS degree in Electronics and Information Engineering from Huazhong University of Science and Technology, China, in 2007. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include distributed networks, with an emphasis on peer-to-peer and content delivery networks, wireless multi-hop cellular networks, game theory and data mining. He is a student member of IEEE.