

Singapore Management University Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

6-2005

DSI: A Fully Distributed Spatial Index for Wireless Data Broadcast

Wang-Chien LEE


Pennsylvania State University

Baihua ZHENG

Singapore Management University, bhzheng@smu.edu.sg

DOI: <https://doi.org/10.1109/ICDCS.2005.26>

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Citation

LEE, Wang-Chien and ZHENG, Baihua. DSI: A Fully Distributed Spatial Index for Wireless Data Broadcast. (2005). *ICDCS 2005: 25th IEEE International Conference on Distributed Computing Systems: Proceedings: 6-10 June 2005, Columbus, Ohio, USA*. 349-358.

Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/522

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

DSI: A Fully Distributed Spatial Index for Location-based Wireless Broadcast Services *

Wang-Chien Lee
The Pennsylvania State University
University Park, PA 16802, USA
wlee@cse.psu.edu

Baihua Zheng
Singapore Management University
Singapore
bhzheng@smu.edu.sg

Abstract

Recent announcement of the MSN Direct Service has demonstrated the feasibility and industrial interest in utilizing wireless broadcast for pervasive information services. To support location-based services in wireless data broadcast systems, a distributed spatial index (called DSI) is proposed in this paper. DSI is highly efficient because it has a linear yet fully distributed structure that facilitates multiple search paths to be naturally mixed together by sharing links. Moreover, DSI is very resilient in error-prone wireless communication environments. Search algorithms for two classical location-based queries, window queries and k NN queries, based on DSI are presented. Performance evaluation of DSI shows that DSI significantly outperforms R-tree and Hilbert Curve Index, two state-of-the-art spatial indexing techniques for wireless data broadcast.

Keywords: Location-based services, spatial index, wireless data broadcast systems

1 Introduction

With the ever growing popularity of smart mobile devices and rapid advent of wireless technology, the vision of *pervasive information access* has come closer to a reality. While information is important to users, it is only valuable when available at the right time, *right place*. The demand for location dependent data (e.g., pollution index, local traffic conditions, restaurant locations, navigation maps, etc.) is tremendous due to the broad application base. Thus, *location* is a very important aspect of pervasive information services.

Pervasive information access via today's wireless technologies (e.g., Bluetooth, IEEE 802.11, UMTS, Satellite) can be captured by two basic approaches: *on-demand access* and *periodic broadcast*. In on-demand access, the

server processes a query and returns query result to the user via a point-to-point channel. On the other hand, periodic broadcast has the server actively pushing data to the users. The server determines the data and its schedule for broadcast. A user listens to a broadcast channel to retrieve data based on its queries and, thus, is responsible for query processing. On-demand access is good for light-loaded systems when contention for wireless channels and server processing is not severe. Broadcast, allowing an arbitrary number of users to access data simultaneously, is suitable for heavy-loaded systems or big events (e.g., the Olympic Games) where the users seek for similar information. Recent announcement of the MSN Direct Service (<http://www.msndirect.com>), which allows mobile devices to receive timely information such as airline schedules, local traffic, weather and news via FM radio subcarrier frequencies, has demonstrated the feasibility and industrial interest in utilizing wireless broadcast for pervasive information services.

Previous research has studied various system issues of wireless data broadcast [2, 3, 4]. In this paper, we address the demand of location dependent data by proposing a novel spatial index, called *Distributed Spatial Index (DSI)*, in support of location-based queries from mobile users in wireless data broadcast systems. The design of DSI has also taken the inherently unreliable and error-prone wireless communication into consideration. DSI is highly efficient due to the following properties: 1) it has a linear structure that fits the wireless broadcast environment very well; 2) it naturally mixes multiple search paths by sharing links among them and thus minimizes the overall storage overhead; 3) it has a fully distributed structure that allows query processing to start very quickly; 4) it is very resilient under error-prone wireless communication environments because interrupted query processing can resume effectively.

Algorithms for processing *window queries* and *k Nearest Neighbor (kNN) queries*, two essential queries for location-based services, on DSI are developed. The issues involved in k NN query processing are particularly challenging. Our

*Wang-Chien Lee was supported in part by US National Science Foundation grant IIS-0328881; Baihua Zheng was supported by the Singapore Management University under grant number 04-C220-LEE-001.

study has led to an innovative solution based on *broadcast reorganization* and resulted in very efficient query processing. Extensive performance evaluation in both error-free and error-prone wireless communication environments has shown that the proposed DSI achieves a significantly better performance than R-tree and Hilbert Curves Index (HCI), two state-of-the-art spatial indexing techniques for wireless data broadcast.

The rest of this paper is organized as follows. Background and existing work to our study is provided in Section 2. The index structure of DSI and algorithms for window and k NN queries are presented in Section 3. Performance evaluation of DSI, HCI, and R-trees indexes for wireless data broadcast is presented in Section 4. Section 5 tests the resilience of these indexes under error-prone wireless environments. Finally, we conclude this paper in Section 6.

2 Preliminaries

Here the background and related work for supporting location-based wireless broadcast services are provided.

2.1 Background

Consider a wireless data broadcast system that periodically broadcasts a collection of data objects to mobile clients. Each data object consists of a set of attribute values. Among them, *location* is particularly important since it is the focus of this paper. *Access efficiency* and *energy conservation* are two critical issues for mobile users, concerning how fast a request could be satisfied and how energy-efficient a technique is. To facilitate energy conservation, a smart mobile device is expected to support two operation modes: *active mode* and *doze mode*. The device normally operates in active mode; and switches to doze mode to save energy when the system becomes idle. In the literature, two performance metrics, namely *access latency* and *tuning time*, are used to measure access efficiency and energy conservation for mobile clients in a wireless data broadcast system, respectively [7, 8, 9]:

- Access Latency: The time elapsed from the moment a query is issued to the moment it is satisfied.
- Tuning Time: The time a mobile client stays active to receive the requested data items.

In wireless data broadcast systems, a client has to stay *active* to continuously receive and check the broadcast data until the data objects of interest arrive. This process consumes a lot of energy. The average tuning time is a half of a *broadcast cycle*, a period when all the available data objects are broadcast once. *Air indexing* is often used for energy conservation at mobile clients [9, 10]. The basic idea is that the broadcast server precomputes indexing information (including indexed attributes and arrival time of data objects) and interleaves it with data objects on the broadcast channel. To facilitate search of data objects via air index, each

data object includes an offset to the next broadcast of index information. As such, mobile clients are able to predict the arrivals of their desired data by examining the index information. Thus, they can stay in *doze mode* most of time and tune into the broadcast channel when the interested data objects arrive.

Many spatial index structures have been proposed for accessing spatial data, including R-tree, KD-tree, Quad-tree, etc [6, 13, 14]. Among those, R-tree is the most well received. A search algorithm based on R-tree typically expands the search space around the query point using a branch-and-bound approach. The navigation order of R-tree is dynamically determined based on the position of the query point, which results in backtracking. Thus, R-tree is better supported by random access storages, such as memory and disk.

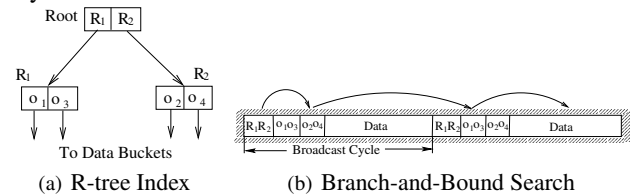


Figure 1. Linear Access on Broadcast Channel

In a wireless broadcast system, however, data objects are broadcast based on a pre-defined sequence (called a *broadcast program*) and thus an object is only available when it is on the air. Consequently, search algorithms designed based on random access may incur a significant access latency. Figure 1 depicts an example. Assume that a search algorithm visits the root node, R_2 and then R_1 , while the server broadcasts nodes in the order of root, R_1 , and R_2 . Consequently, if a client wants to visit node R_1 after it retrieves R_2 , it will have to wait until the next cycle because R_1 has already been broadcast. This significantly extends the access latency and it occurs every time a search order is different from the broadcast order. To address this issue, the navigation order of R-tree needs to follow the broadcast order of index nodes.

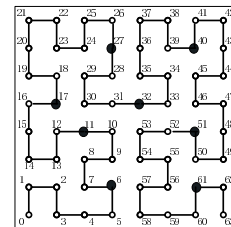


Figure 2. Hilbert Curve of Order 3

Since most location-based queries search for objects located closely, one strategy for wireless data broadcast is to schedule spatially near objects to broadcast close to each other. To achieve this, an idea is to broadcast data objects based on a space-filling curve (e.g., *Hilbert Curves* (HC)

[5]) and build an air index upon. The key is to keep neighbors in a high dimensional space remaining close to each others in the broadcast channel. Figure 2 shows an HC of order 3. The numeric labels represent the positions of the objects in terms of *HC values*. For instance, point (1,1) has the HC value of 2. The order of the curve is decided by the object distribution in the space. HC of higher order is needed for denser object distribution since the curve has to pass through all the objects. There is a 1-1 correspondence between the coordinate and HC value of a data object. Given the mapping function of HC, it is easy for a client to perform conversion between coordinates and HC values in a constant time. The detailed conversion algorithm is available in [12].

2.2 Related Work

Imielinski et al. has extended B^+ -tree to support the access of broadcast data. Two approaches are proposed to interleave the index and data in a broadcast channel, namely $(1, m)$ and *distributed index* [9]. The former treats the whole index as a segment and replicates the index segment m times during one broadcast cycle. Thus, the clients suffer a longer access latency since the m duplicated index segments extend the overall broadcast cycle. The distributed index replicates only the top part of an index tree (see Figure 3 for illustration). It has been shown that the distributed index scheme is more efficient than $(1, m)$ in terms of access latency.

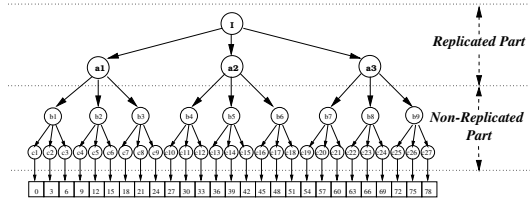


Figure 3. Distributed Index for Broadcast

Ideas of indexing the attribute ranges of exponentially increasing number of data objects were discussed in the literature, e.g., *Chord* [15], *flexible index* [8] and *exponential index* [16]. However, the focus of these work is totally different from our study. Chord aims at providing peer-to-peer lookup based on a hashed search key, while flexible index and exponential index investigate optimal tuning of the access latency and tuning time in support of simple search of broadcast data based on *single* attribute. None of them considered complex location-based queries as we do in this paper.

Some air indexes have been recently proposed to support broadcast of location dependent data [17, 18, 19]. Among them, the *D-tree* is a paged binary search tree to index a given solution space in support of planar point queries [17], while the *Grid-partition index* is specialized for the (one) nearest neighbor problem [19]. *Hilbert Curve Index (HCI)*

is designed to support both of window queries and nearest-neighbor queries at the same time. It adopts a B^+ -tree to index data objects broadcast according to the Hilbert Curve order [18]. The HCI index structure and search algorithms are totally different from DSI. Also worthnoting is that the reorganized DSI data broadcast *does not* follow HC order.

3 Distributed Spatial Index (DSI)

All the spatial air indexes reviewed earlier are based on tree structures. Thus, a search always starts at the root, which results in some deficiencies. First, the clients have to wait for the arrival of the root node to start the search. Moreover, the search has to be stopped once an index node along the search path is lost. The client must wait for the next root or blindly scan all the following nodes in order to resume the search. Therefore, those indexes only perform well under an ideal situation where no packet loss happens.

To address these inherited deficiencies of tree indexes on air, we propose a fully distributed spatial index structure, namely *DSI*, to allow a client to start query processing as soon as possible in order to minimize the access latency while still conserving tuning time. DSI distributes the index information over the whole broadcast cycle and equips a client, whenever it tunes into the channel, with sufficient information to conduct the location-based search.

In the following, we introduce the index structure of DSI, energy efficient forwarding, and the search algorithms for window query and k -Nearest-Neighbor (k NN) query. A technical challenge which arises in k NN search is discussed and a refined broadcast organization is proposed to further improve the search performance.

3.1 The Index Structure

Taking into account the sequential access property of wireless broadcast, Hilbert Curve (HC) is adopted in the initial design of DSI to determine broadcast order of data objects. By default, data objects are broadcast in the ascending order of their HC values. The basic idea is to divide the whole set of data objects into n_F frames and associate with each frame an *index table*. The index table maintains information regarding to the HC values of data objects to be broadcast with specific waiting interval from the current frame¹.

A DSI index table consists of a number of table entries, τ_i , in the form $\langle HC_i^r, P_i \rangle$, where $0 \leq i \leq (\log_r(n_F) - 1)$, r is a selected exponential base (called *index base*), and n_F is the number of the frames within one broadcast cycle. Note that, logically, the set of frames starting from any arbitrary frame F until the frame before reappearance of F forms a broadcast cycle. Therefore, the index table associated with a frame F is designed to cover the next $(n_F - 1)$ frames following F , i.e., providing index information on HC val-

¹This interval can be denoted by time or by number of data packets.

ues of all the frames within a broadcast cycle. P_i points to the next r^i th frame. HC'_i is the smallest HC value of the objects within the frame pointed by P_i . Thus, the i th entry of the index table provides range information of HC values amongst data objects in the r^i th to $(r^{i+1} - 1)$ th frames. In other words, the number of frames covered is exponentially increased with the order of index table entries.

The number of data objects within a frame n_o (called *object factor*) can be decided based on various parameters such as the size of data packets, etc. Object factor is correlated to the overall index size and the average waiting time to reach the next index table. Additionally, the index base r can be chosen to control the overhead of index table, i.e. the number of entries within one index table. For simplicity, we assume the object factor to be 1 and the index base to be 2 in our discussion. Figure 4 shows the broadcast of data objects based on the example in Figure 2 and the index tables. In this example, $n_F = 8$ and thus each index table has 3 entries. The DSI tables corresponding to frames of data objects O_6 and O_{32} are also shown in the figure. Take the index table for frame O_6 as an example: τ_0 contains a pointer to the next upcoming frame whose HC value is 11, τ_1 contains a pointer to the second frame with HC value 17, and the last entry τ_2 points to the fourth frame with HC value 32.

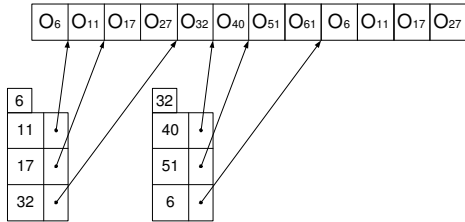


Figure 4. DSI for the Running Example

3.2 Energy Efficient Forwarding

One essential operation for DSI, called *energy efficient forwarding* (EEF), is to efficiently reach a frame containing the data object of a given location. This operation becomes *point query* if the reached frame is scanned to find the queried data object. The EEF algorithm based on DSI is informally given as follows.

Given a target point p , a client first computes the HC value of point p , denoted as HC_p , and then tunes into the broadcast channel. After the initial probe, it downloads the index table associated with the first frame encountered. By comparing HC_p with HC'_i s maintained in the index table, the client follows the pointer P_i , where $HC_p \in [HC'_i, HC'_{i+1})$. In other words, the client goes into the doze mode and wakes up when the frame specified by P_i arrives. This process continues until the frame containing data object located at p is reached. As shown earlier, DSI organizes the index information in a way such that the ranges of HC values amongst exponentially increasing seg-

ments of frames are maintained in the index tables. As a result, EEF is logically like a binary search (when the index base is 2) and the distances between visited frames and the final target frame decrease rapidly.

3.3 Window Queries

A *window query* returns all the data objects associated with locations within a given query window W (e.g., a rectangle in a two-dimensional space). To process a window query with DSI, all segments along the HC located within W (called *target segments*) are found. Since HC is linear, a target segment seg within W must share some common endpoints with segments outside W . All these common endpoints must be on the boundary of W . Hence, the window query algorithm first detects all the intersections between the HC and the boundary of W . Without loss of generality, we assume that all the segments located inside the query window form a target segments set H . Upon receiving an index table, the client sequentially scans each entry and follows the first pointer P_i with the range $[HC'_i, HC'_{i+1})$ overlapping with some segment seg_j of H . The EEF algorithm is then invoked to reach the first frame on seg_j to retrieve data objects. Thereafter, the segment seg_j is removed from H . This process continues until H is empty. As shown, EEF allows clients to move from one target segment to the next one efficiently. The detailed window query algorithm is provided in Algorithm 1.

Algorithm 1 Window Query

Input: a query window w ;
Output: objects within query window;
Procedure:

- 1: compute the target segments set H , with each seg_i denoted by $[H_{2i-1}, H_{2i}]$;
- 2: begin the initial probe and retrieve frame F_s ; $result = \emptyset$;
- 3: let seg_i be the target segment closest to F_s ;
- 4: $F_s = EEF(H_{2i-1})$;
- 5: **while** H is not empty **do**
- 6: **for** each object O covered by one target segment **do**
- 7: $result = result \cup \{O\}$;
- 8: **end for**
- 9: $H = H - [H_{2i-1}, H_{2i}]$;
- 10: scan index table and let HC'_i be the first HC value covered by H
- 11: $F_s = EEF(HC'_i)$
- 12: **end while**
- 13: return $result$;

Figure 5 illustrates the window query processing with DSI. Suppose the shaded area in the figure is a query window. The client first has to detect all the target segments. In this example, the set H has three target segments, $[10, 11]$, $[28, 35]$, and $[52, 53]$. Suppose the client tunes into the channel as depicted, the first frame F_1 it receives contains object O_6 , which is not located within the query window. The client then follows the first index entry to retrieve O_{11} and removes the target segment $[10, 11]$ from H . By combining index tables in F_1 and F_2 , the client has the knowl-

edge of most of the objects, i.e., (6, 11, 17, 27, 32, 40). Thus, she skips F_3 and F_4 by going to doze mode (as indicated by dark frames) and only wakes up to retrieve F_5 . At this moment, the search still does not terminate since there may be objects within segment [52, 53]. After receiving F_7 , the search terminates since O_{52} and O_{53} do not exist.

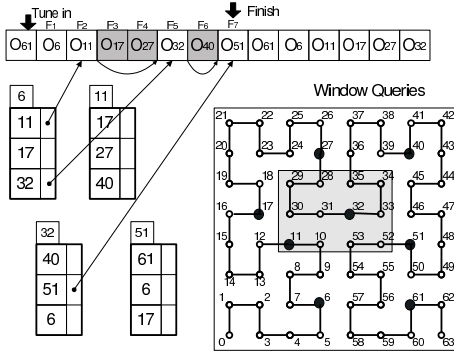


Figure 5. Window Query Processing

3.4 k Nearest Neighbor Queries

A k nearest neighbor (kNN) query finds the nearest k objects to a query point. The basic idea behind our kNN algorithms is to determine a search space based on the partial knowledge of object distribution obtained from index table. The search space will continuously shrink as more knowledge of the data distribution is obtained.

The challenge is to quickly determine a *precise* search space which contains all the k objects. The initial search space is *the whole spatial region* covering all the data objects in the system. As a client tunes into the broadcast channel to receive the first index table, a circle centered at the query point (which specifies a search space) is drawn to include at least k data objects. If the query point is located far away from the current broadcast frame, the circle could be very large because the index table has very limited information about distribution of data objects far away (due to exponential increase of data objects covered by index table entries). As the client continues to monitor the broadcast channel and to obtain more information about object distribution from index tables of subsequently broadcast frames, the circle can be shrunk to avoid retrieval of frames containing unwanted objects. On the other hand, if a query point is close to the current broadcast frame, the search space will converge very rapidly and the search process typically will terminate quickly, because there are more index entries in DSI covering data objects to be broadcast soon. The search space is finalized when no more objects could further reduce the radius. The search is completed when all the objects within the search space are retrieved. Based on how to determine the search space, two strategies for kNN query processing, namely *conservative* and *aggressive*, are described.

Conservative Approach. This approach retrieves a data object if it may potentially be in the answer set. Thus, a client tends to retrieve a lot of subsequent frames and use their associated index tables to reduce the search space. This simple approach has small access latency but suffers from the high energy expense due to slow search space convergence. Instead of going into detailed algorithm (see Algorithm 2), we use a simple example (shown in Figure 6) to illustrate the conservative kNN query processing.

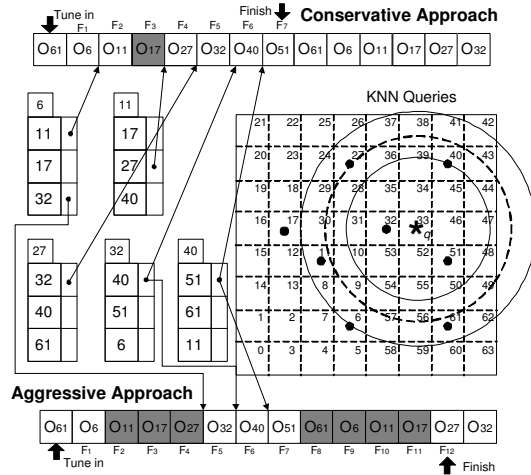


Figure 6. kNN Query Processing

A client physically located at the spot labelled by HC value 33 would like to find 3 nearest neighbors (e.g., restaurants) and tunes into the channel as depicted. From the index table of F_1 , the client knows the existence of objects O_6 , O_{11} , O_{17} , and O_{32} . Accordingly, the client can determine that objects O_6 , O_{11} , O_{32} are the three nearest neighbors she knows so far (among them, O_6 located farthest from the query point). Thus, a solid circle across O_6 is determined as the search space (see Figure 6). After retrieving object O_6 , the client follows the first pointer to access F_2 , and detects the existence of objects O_{27} and O_{40} . Consequently, the search space is further reduced to the dashed circle across O_{27} . The client ignores object O_{11} and skips download of frame F_3 (since they are outside of the current search circle), and then accesses F_4 directly. The process of refining search space continues and only qualified objects are retrieved. Eventually, the search space will stop shrinking (i.e. the inner solid circle) and all the searched objects within the space (i.e., O_{32} , O_{40} and O_{51}) are retrieved. The search process is terminated after O_{51} is retrieved because no object in the following broadcast frame will be within the search range. As Figure 6 shows (dark frame means the client is in doze mode), the access latency is 7 frames and the tuning time is 6 frames.

Aggressive Approach. This approach uses a different strategy in determining the next frame to retrieve. As discussed earlier, the index tables associated with frames far away

Algorithm 2 Conservative k NN query processing

Input: a query point, q , and the number of nearest neighbors, k ;
Output: k nearest neighbors to q ;
Function: $\text{insert}(\text{result}, \text{object}, k)$ - insert object into result and maintain only the k nearest candidates in result ; finally return the maximal distance between query point and the candidates.

end insert
Procedure:
 1: $r = \infty$; $\text{res} = \emptyset$; $F' = \text{NULL}$;
 2: begin the initial probe and retrieve frame F_s ;
 3: **while** F_s **do**
 4: **for** all the pointers P_i in the index table of F_s **do**
 5: o'_i = the object represented by HC'_i ;
 6: **if** $\text{dis}(o'_i, q) < r$ **then**
 7: $r = \text{insert}(\text{res}, o'_i, k)$;
 8: **end if**
 9: **end for**
 10: **for** all the data objects obj_i within F_s **do**
 11: **if** $\text{distance}(obj_i, q) \leq r$ **then**
 12: retrieve obj_i ; $r = \text{insert}(\text{res}, obj_i, k)$;
 13: **end if**
 14: **end for**
 15: **for** all the pointers P_i in the index table of F_s **do**
 16: F_p = the frame pointed by the pointer P_i ;
 17: **if** $\text{dis}(o', q) \leq r$ with $HC_{o'} \in [HC'_i, HC'_{i+1})$ **then**
 18: $F' = F_p$; **break**;
 19: **end if**
 20: **end for**
 21: $F_s = F'$; $F' = \text{NULL}$;
 22: **end while**
 23: **return** res ;

from the query point usually do not provide good knowledge about data distribution that we are interested in. It will be wise to access index tables closer to the query point in order to shrink the search space more rapidly and to skip data objects which eventually fall out of the search space. Thus, the aggressive approach follows an index entry which points to a frame closest to the query point. The advantage of this approach is fast convergence of the search space and energy efficiency. However, it may possibly skip some queried objects and have to wait until the next broadcast cycle to retrieve them.

Since the algorithm is very similar to the conservative approach, it is not listed in the paper to save space. Instead, the same running example is used to illustrate the aggressive k NN processing (also shown in Figure 6). Based on the index table of F_1 , the client knows the objects distribution is like (6, 11, 17, ?, 32, ?, ?, ?), where ? represents the unknown objects. Thus, the solid circle across O_6 is drawn to obtain the search space. Next, it follows the third pointer to retrieve F_5 , the reachable frame nearest to the query point. As a result, frames F_2 , F_3 , and F_4 are skipped (i.e., the client turns to doze mode as indicated by dark frames). According to the index table of F_5 , the client has more knowledge of the objects distribution, i.e. (6, 11, 17, ?, 32, 40, 51, ?). Since objects O_{32}, O_{40}, O_{51} are the three nearest neighbors the client knows so far, a circle across O_{40} (i.e.,

the inner solid circle) is drawn as the search space. Since O_{32} is right in front of the query point (i.e., 33), the following search steps are similar to those in the conservative approach, i.e., sequentially retrieving all the data objects located within the search space. After retrieving O_{51} , however, the search process cannot be terminated yet since the HC range 11..31 has been skipped earlier (note that $O_{7..O_{10}}$ have been known to be non-existing back then). As a result, O_{28} and O_{31} may still exist in the converged search space and thus need to be checked. Finally, the client goes into doze mode to skip F_8 to F_{11} and then wakes up to retrieve O_{27} (i.e., F_{12}). The index table shows the next object is O_{32} , ruling out the existence of O_{28} and O_{31} , and thus terminates the search process. As Figure 6 shows, the access latency is 12 frames and the tuning time is 5 frames.

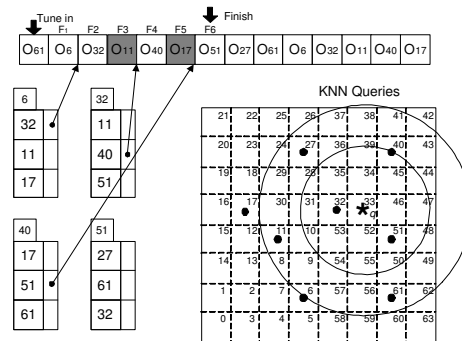


Figure 7. Broadcast Reorganization

3.5 Broadcast Reorganization

DSI tables allocate more index entries on *nearby* frames (i.e., those to be broadcast soon) and fewer index entries on *remote* frames which are still far away. Thus, the search space of a k NN query determined from low precision object distribution information of remote frames usually is excessively large. To avoid missing any qualified data object, the conservative approach retrieves all the objects within the current search space. Thus, it may waste a significant amount of energy for retrieving a lot of data objects which may eventually fall out of the final answer set, like objects O_6 and O_{27} in the previous example. Even though additional index information is obtained when new frames are retrieved, it may not provide significant help to the conservative approach because the index tables close to each other usually have a lot of overlapped coverage on object distribution. On the contrary, aggressive approach skips a lot of frames (by tuning into doze mode) to get closer to the query point. Hence, the aggressive approach converges the search space faster and retrieves less unqualified objects (i.e., it is very energy efficient). A drawback for this approach is the longer access latency needed to terminate the search. For example, frames F_2 to F_4 are skipped in the aggressive approach, which extends the access latency to nearly one and a half broadcast cycle. Thus, the conservative and aggres-

sive approaches represent a tradeoff between access latency and energy efficiency.

With the current HC order of data broadcast in DSI, neither conservative nor aggressive approach can minimize access latency and tuning time at the same time. To address this issue, a broadcast reorganization is proposed to facilitate retrieval of more accurate object distribution information about remote frames. The basic idea is to reorganize the broadcast of data objects in an order other than linear ascending to allow the clients to receive index information about remote object distribution without missing qualified data objects.

The reorganization of data broadcast is simple yet effective. Suppose a broadcast cycle is based on the ascending order of objects' HC values. The broadcast cycle can be evenly divided into m broadcast segments with the same number of frames. Next, the broadcast cycle is reconstructed by interleaving frames from these m segments. Figure 7 shows a broadcast cycle reorganized by interleaving two broadcast segments (i.e., $m = 2$). The advantage of this broadcast reorganization is that the search space of a k NN query can be quickly reduced by jumping into a remote area (i.e., a different segment) closer to the query point because higher precision object distribution surrounding the query point can be obtained. One can still come back to the original segment without losing desired data objects. This allows a mobile client to obtain different views of the object distribution from different broadcast segments and thus quickly shrink the search space. As the running example in Figure 7 shows, the two-segment broadcast reorganization (with the conservative search strategy) improves both the access latency (4 frames) and tuning time (6 frames).

4 Performance Evaluation

This section evaluates the performance of DSI by comparing it with R-tree and Hilbert Curve Index (HCI). Two datasets, UNIFORM and REAL, are used in the evaluation. In the UNIFORM dataset, 10,000 points are uniformly generated in a square Euclidean space. The REAL dataset contains 5848 cities and villages in Greece, extracted from the point dataset available in [1]. Since data objects are available a priori, the STR packing scheme is employed to build R-tree in order to provide an optimal performance [11]. On the other hand, B⁺-tree is used in HCI to index data objects broadcast in linear order of Hilbert Curve. Thus, they are denoted as *R-tree* and *HCI* in our discussions. Both implementation of R-tree and B⁺-tree are based on the well known distributed indexing scheme [9].

DSI can be constructed based on different configurations of exponential base r and object factor n_o . For simplicity, r is fixed to 2 in our simulation. There is a relationship between n_o and n_F , i.e., the total number of data objects N equals to $n_F \cdot n_o$. To determine the object factor n_o , we

allocate one packet for each index table associated with a frame. Thus, the number of table entries that could fit into one packet, i.e. $\log_2(n_F)$, is obtained and the total number of frames, n_F , can also be derived. DSI is thereafter constructed based on n_F and n_o .

The simulation model consists of a base station, an arbitrary number of clients, and a broadcast channel. The *WinSideRatio* for window query is defined as $\frac{\text{the side length of query window}}{\text{the side length of the whole search space}}$ (default = 0.1). The packet size, denoted as *Capacity*, varies from 2⁵ bytes to 2⁹ bytes in the experiments (default = 64 bytes). The size of a data object is set to 1024 bytes. A two-dimensional coordinate is represented by two floating-point numbers (8 bytes each) and the HC value is represented in the same total size (16 bytes). For each pointer in the index table, 2 bytes are allocated. We use *total bytes* instead of *number of packets* to measure access latency and tuning time because the packet capacity is varied in many experiments². Showing results in terms of number of packets in figures does not reveal much insight between implementations based on different packet sizes. Also note that R-tree is not implemented with packet size of 32 bytes because more space is needed to maintain MBR and pointers in index nodes. This is a limitation of R-tree which does not exist in DSI and HCI. Thus, the figures involving R-trees do not include performance result under packet capacity of 32 bytes. Due to space constraint, we only present in figures the simulation results using UNIFORM dataset. The results obtained using REAL dataset are summarized in our discussions.

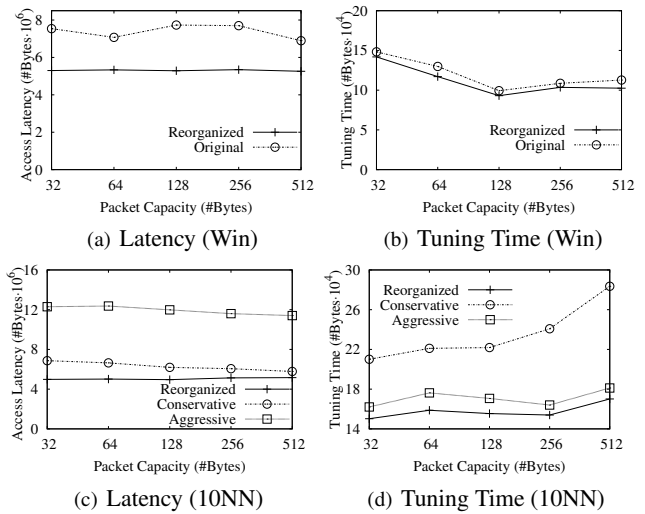


Figure 8. Broadcast Reorganization (UNIFORM)

4.1 Broadcast Reorganization

DSI does not necessarily follow HC order! In this paper, we propose to reorganize the broadcast order of data ob-

²With a known packet capacity, conversion between the number of packets and total bytes is straightforward.

jects in order to address issues arise in k NN query processing. Here we validate our proposal of broadcast reorganization under both window and k NN queries. We adopt a two-segment broadcast reorganization in the experiment and compare it with the original broadcast based on the ascending order of HC values. Figure 8 shows the performance of DSI with or without object reorganization under the UNIFORM dataset. For k NN query, both conservative and aggressive algorithms of the original broadcast are included. The curve labelling in the figures is self-explained.

The improvement of access latency made by broadcast reorganization on window queries is quite evident, reducing an average (over all packet capacities) of 28% access latency over the original broadcast program. It also improves the tuning time performance, with an improvement of around 7% (average over all packet capacities). We also observe that the access latency is not affected by packet capacity, while the optimal tuning time performance is reached when packet capacity = 128 bytes (see Fig. 8(a)-(b)). The performance enhancement on window queries is due to more flexible access to different broadcast segments facilitated by broadcast reorganization. On the other hand, the improvement of the broadcast reorganization on k NN queries is even more astonishing (in both of access latency and tuning time). Fig. 8(c)-(d) show that the conservative approach is good for access latency while the aggressive approach can save tuning time. Nevertheless, the broadcast reorganization is the best. It allows a client to access useful distribution information of remote objects near a query point early such that search space converges quickly (like the aggressive approach). Meanwhile, data objects belonging to the answer set are not missed (like the conservative approach). For the rest of experiments, we employ reorganized broadcast for DSI.

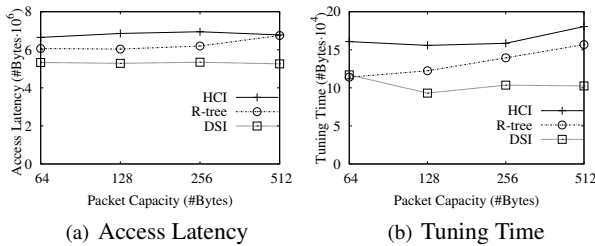


Figure 9. Performance of Window Queries vs. Packet Capacities (UNIFORM)

4.2 Window Queries

Here we compare the performance of window query of the evaluated indexes. We first fix the size of query window and vary packet capacity from 32 to 512 bytes to evaluate the performance of window query processing (the points corresponding to packet capacity=32 bytes are not shown due to the constraint of R-tree). Figure 9 shows that DSI is superior to R-tree and HCI. As the packet capacity in-

creases, the performance of DSI remains stable while the access latency and tuning time of R-tree and HCI both increase. On average, DSI requires only 85.4% of R-tree latency and 77.7% of HCI latency under UNIFORM dataset. The advantage of DSI under REAL dataset (not shown to save space) is even more significant, requiring only 59.7% of R-tree latency and merely 50.5% of HCI latency, respectively. DSI also performs very well in terms of tuning time. On average, it consumes only 79.6% tuning time of R-tree and 63.7% tuning time of HCI under UNIFORM dataset. Meanwhile, it consumes only 75.2% tuning time of R-tree and 41.5% tuning time of HCI under REAL dataset (not shown to save space).

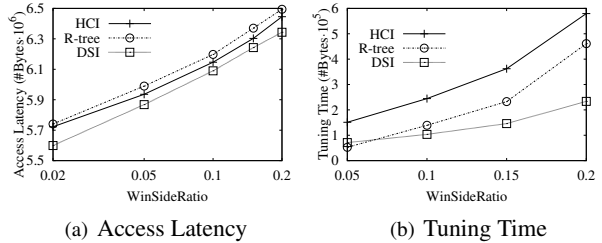


Figure 10. Performance of Window Queries vs. WinSideRatio (UNIFORM)

In order to provide a comprehensive evaluation, we next fix packet capacity to 64 bytes but vary query window sizes by changing *WinSideRatio*. Figure 10 shows that DSI outperforms R-tree and HCI in general. The simulation using REAL dataset has similar result. As expected, access latency and tuning time for all indexes increase as the window size increases. DSI does not perform as well as R-tree (in terms of tuning time) only when the window size is very small. This is because R-tree ensures high spatial locality. When the query window is very small, it needs to access less number of leaf nodes, and hence has a better performance. On the other hand, while DSI maintains a good degree of spatial locality through HC, a small query window does not necessarily imply a small HC range to search.

4.3 K Nearest Neighbor Queries

In this section, the performance of k NN queries under different indexes is evaluated. We first examine NN and 10NN queries by varying packet capacity. Figure 11 plots the performance of evaluated indexes using the UNIFORM dataset. The result using REAL dataset has similar conclusion and thus not included. DSI outperforms both R-tree and HCI significantly. On average, DSI incurs only 23.3% of HCI and 58.6% of R-tree access latency, respectively, and DSI consumes 26.9% HCI and 41.7% R-tree tuning time, respectively (for NN search). When k increases to 10, the advantage of DSI is still very significant. On average, DSI incurs 25.2% of HCI and 65.1% of R-tree access latency, respectively (for 10NN search). Meanwhile, DSI consumes 37.6% HCI and 31.8% R-tree tuning time (for

10NN search).

This experiment shows that the performance of DSI is very stable under various packet capacities. The access time is very bounded by broadcast cycle and thus does not change much. As for the tuning time, when the packet capacity increases, DSI increases only slightly while HCI and R-tree increase very rapidly. Compared with window queries, the advantages of DSI are more dramatic for k NN queries. Broadcast reorganization benefits our search strategy of refining the search space gradually while advancing close to the query point. Hence, the performance is significantly improved.

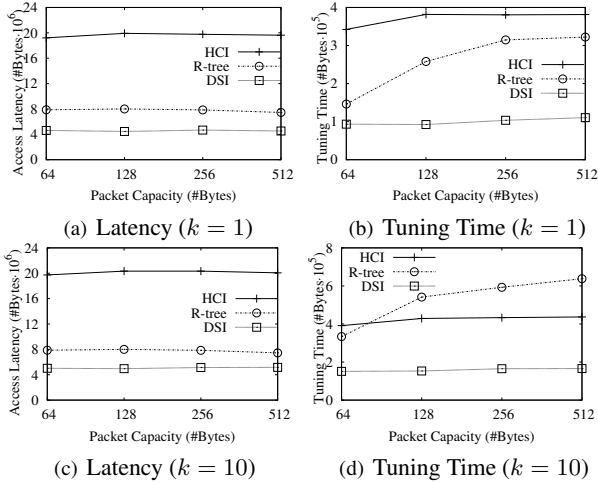


Figure 11. Performance of k NN Queries Under Various Packet Capacity (UNIFORM)

Next, we examine the impact of the number of searched nearest neighbors on the performance of spatial air indexes. Figure 12 shows the performance of different indexes under different settings of k for the UNIFORM dataset. Again, the result using REAL dataset has similar conclusion and thus not included. As expected, DSI performs the best in all the cases, in terms of both access latency and tuning time. As k increases, the access latency, bounded by the size of a broadcast cycle, does not change much. On the other hand, the tuning time of DSI increases in a much slower speed than that of R-tree and HCI.

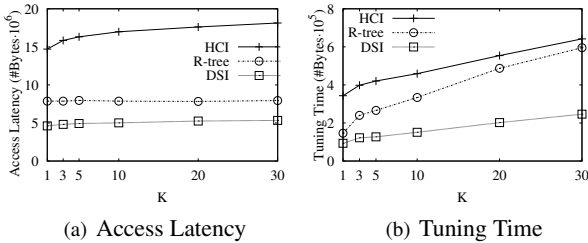


Figure 12. Performance of k -NN Queries vs. K (UNIFORM)

5 Resilience to Link Errors

The performance of different indexes presented earlier is based on a reliable wireless communication environment, which assumes no interference and packet loss. However, wireless environment is inherently unreliable and error prone due to radio propagation attenuation, fading and noise. In such an environment, link errors occur frequently and thus an air indexing technique which can facilitate recovery of query processing after interruption is highly desirable. DSI is able to handle link errors easily owing to its fully distributed index structure. There are multiple search paths to reach a destination (frame). If a frame is corrupted, the client can easily resume the query processing in the next frame while continue to use the knowledge of data distribution obtained previously. Thus, the performance penalty is minimized. This is a great advantage of DSI.

For a tree index, any node is only pointed by one parent. Therefore, the immediate access to this node will be lost if its parent node can not be reached. The client either has to wait for rebroadcast of the lost parent node or blindly retrieves all the following packets from the wireless channel until the desired node is received. However, both approaches are inefficient. The former incurs an extremely large access latency since the loss of any node along the search path will interrupt the search. The latter approach wastes scarce power resources by scanning lots of useless nodes.

Index	θ	Window Query		10NN	
		Latency	Tuning	Latency	Tuning
HCI	0.2	3.02%	0.97%	11.64%	6.00%
	0.5	11.70%	4.76%	23.65%	9.27%
	0.7	28.96%	12.07%	34.99%	18.13%
R-tree	0.2	7.50%	1.10%	11.48%	6.19%
	0.5	21.21%	5.27%	27.47%	12.84%
	0.7	62.40%	14.12%	59.26%	18.16%
DSI	0.2	0.70%	0.88%	6.66%	3.93%
	0.5	5.19%	3.71%	20.12%	7.16%
	0.7	13.90%	8.03%	30.45%	10.41%

Table 1. Performance Deterioration in Error-Prone Environments (UNIFORM)

The above analysis points out the strength of DSI and deficiency of tree indexes in an error-prone environment. Next, we evaluate the resilience of air spatial indexing techniques to link errors quantitatively. We use a variable θ to control the percentage of link errors in the broadcast system. When θ is 0, there is no packet loss. When θ is 1, all packets are lost. The performance of indexes under different ratios of link errors is evaluated. The value of θ is varied from 0, 0.2, and 0.5, to 0.7. Again, DSI shows a superior performance in all scenarios. Table 1 summarizes the performance deterioration (in percentage) under various link error ratios in comparison with the same indexes in ideal,

lossless communication environment (i.e., $\theta = 0$). Note that DSI has the smallest performance deterioration and the best baseline performance (as shown by experimental results in Section 4), which really showcases its resilience to error-prone wireless communication environments.

6 Conclusion

A desirable air indexing technique should exploit properties of wireless broadcast, facilitate access and energy efficient query processing, and be resilient in error-prone wireless environments. In this paper, a fully distributed spatial index, named *DSI*, is proposed to address these requirements. DSI naturally mixes multiple search paths into a linear index structure which is fully distributed into the whole broadcast cycle. As a result, it allows a search to start immediately and facilitates instant recovery of interrupted query processing when a packet is corrupted or lost.

This study significantly contributes to the development of location-based wireless data broadcast services and the research community in the following ways: 1) many insights on location-based wireless broadcast and query processing are obtained; these insights lead to the design of this new and very novel DSI air index; 2) new algorithms for window queries and k nearest neighbor search (based on different strategies) are developed; 3) a broadcast reorganization technique is proposed for enhancement of DSI based on obtained insights; 4) extensive (simulation-based) experiments have been conducted to compare the performance of DSI with R-trees and HCI (two state-of-the-art techniques) in both of error-free and error-prone wireless environments. The result shows the superiority of DSI in both environments (details were shown earlier).

As for the future work, we are currently looking into prototyping of a location-based wireless broadcast system. Meanwhile, we continue to exploit the potential impacts and use of broadcast reorganization and the research issues in unreliable wireless communication environments.

References

- [1] Spatial datasets. Website at <http://www.rtreeportal.org/spatial.html>.
- [2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 199–210, San Jose, CA, USA, May 1995.
- [3] A. Datta, A. Celik, J. Kim, D. VanderMeer, and V. Kumar. Adaptive broadcast protocols to support power conservation retrieval by mobile users. In *Proceedings of IEEE International Conference Data engineering*, pages 124–133, Birmingham, UK, April 1997.
- [4] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast protocols to support efficient retrieval from databases by mobile users. *ACM Transactions on Database Systems (TODS)*, 24(1):1–79, March 1999.
- [5] C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing*, 5(5):794–797, May 1996.
- [6] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 47–54, 1984.
- [7] Q. L. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'2000)*, pages 157–166, San Diego, CA, USA, February 2000.
- [8] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficiency filtering of data on air. In *Proceedings of the 4th International Conference on Extending Database Technology (EDBT'94)*, pages 245–258, Cambridge, UK, March 1994.
- [9] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air - organization and access. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(3), May-June 1997.
- [10] W.-C. Lee and D. L. Lee. Using signature techniques for information filtering in wireless and mobile environments. *Journal of Distributed and Parallel Databases (DPDB)*, 4(3):205–227, July 1996.
- [11] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. Str: A simple and efficient algorithm for r-tree packing. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 497–506, Birmingham, UK, April 1997.
- [12] D. Moore. Hilbert Curve. URL at <http://www.caam.rice.edu/~dougm/twiddle/Hilbert>.
- [13] J. Robinson. The kdb tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, Ann Arbor, Michigan, 1981.
- [14] H. Samet. The quad tree and related hierarchical data structures. *ACM Computing Surveys*, 16(2), 1984.
- [15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, August 2001.
- [16] J. Xu, W.-C. Lee, and X. Tang. Exponential index: A parameterized distributed indexing scheme for data on air. In *Proceedings of the 2nd ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys '04)*, Boston, MA, June 2004.
- [17] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee. Energy efficient index for querying location-dependent data in mobile broadcast environments. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03)*, Bangalore, India, March 2003.
- [18] B. Zheng, W. C. Lee, and D. L. Lee. Spatial index on air. In *Proceedings of the first IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, Dallas-Fort Worth, Texas, USA, March 2003.
- [19] B. Zheng, J. Xu, W. C. Lee, and D. L. Lee. Energy-conserving air indexes for nearest neighbor search. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT'04)*, Heraklion - Crete, Greece, March 2004.