8-2009

A Distributed Spatial Index for Error-Prone Wireless Data Broadcast

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Wang-Chien LEE
Pennsylvania State University

Ken C. K. LEE
Pennsylvania State University

Dik Lun LEE
Hong Kong University of Science and Technology

Min SHAO
Pennsylvania State University

Noname manuscript No.
(will be inserted by the editor)

A Distributed Spatial Index for Error-Prone Wireless Data Broadcast

Baihua Zheng[1], Wang-Chien Lee[2], Ken C. K. Lee[2], Dik Lun Lee[3], Min Shao[2]

[1] Singapore Management University, Singapore. bhzheng@smu.edu.sg
[2] Pennsylvania State University, USA. {wlee,cklee,mshao}@cse.psu.edu
[3] The Hong Kong University of Science and Technology, Hong Kong. dlee@cse.ust.hk

Abstract Information is valuable to users when it is available not only at the right time but also at the *right place*. To support efficient location-based data access in wireless data broadcast systems, a distributed spatial index (called *DSI*) is presented in this paper. DSI is *highly efficient* because it has a linear yet fully distributed structure that naturally shares links in different search paths. DSI is *very resilient* to the error-prone wireless communication environment because interrupted search operations based on DSI can be resumed easily. It supports search algorithms for classical location-based queries such as window queries and kNN queries in both of the *snapshot* and *continuous* query modes. In-depth analysis and simulation-based evaluation have been conducted. The results show that DSI significantly out-performs a variant of R-trees tailored for wireless data broadcast environments.

Keywords: mobile computing, location-based query, wireless broadcast, error resilience

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 reality. While information is important to users, it is valuable only when available at the right time, *right place*. Indeed, *location* is a very important requirement to pervasive information access. The demand for access of location dependent data (e.g., pollution index, local traffic conditions, restaurant locations, navigation maps, weather condition, etc.) fosters a tremendous application base of location based services.

Today, there are many wireless technologies (e.g., Bluetooth, WiFi, UMTS, Satellite, etc.) that could be integrated into a seamless, pervasive information access platform. Logically, information access via these wireless technologies can be classified into two basic approaches: *on-demand access* and *periodic broadcast*. On-demand access employs a pull-based approach where a mobile client initiates a query to the server which in turn processes the query and returns the result to the client over a point-to-point channel. On-demand access is suitable for lightly loaded systems in which wireless channels and server processing capacity is not severely contended.

On the other hand, periodic broadcast requires the server to proactively push data to the clients over a dedicated broadcast channel. This approach allows an arbitrary number of clients to access data simultaneously, and thus is particularly suitable for heavily loaded systems. Wireless data broadcast services have been available as commercial products for many years, e.g. StarBand (www.starband.com) and Hughes Network (www.hughesnet.com). Recently, there has been a push for such systems from the industry and various standard bodies. For example, born out of the International Telecommunication Union's (ITU) International Mobile Telecommunications "IMT-2000" initiative, the Third Generation Partnership Project 2 (www.3gpp2.org) is developing Broadcast and Multicast Service in cdma2000 Wireless IP network [36]. Another example is the MSN Direct Service (www.msndirect.com) based on the smart personal objects technology (SPOT) and the DirectBand Network. With a continuous broadcast network using FM radio subcarrier frequencies, mobile devices (e.g., smart watches, Navigators and PDAs) can continuously receive timely information such as stock quotes, airline schedules, local news, weather, and traffic information. Moreover, systems for Digital Audio Broadcast (DAB) (www.worlddab.org) and Digital Video Broadcast (DVB) (www.dvb.org) are capable of delivering wireless data services. News also reported that XM Satellite Radio (www.xmradio.com) and Raytheon have jointly built a communication system that would use XM's satellites to relay information to soldiers and emergency responders during a homeland security crisis.

A number of studies have addressed various system issues of wireless data broadcast [3,7,8,16,18,31,41]. Recently, several proposals on delivering location-based data via wireless data broadcast have been suggested, e.g., enhanced R-tree based air index by Gedik, et. al. [12], and the DAYS project led by Kumar [2,10]. In this paper, we address the demand of location-based data by proposing a novel on-air spatial index, called *Distributed Spatial Index (DSI)*, to support location-based queries issued by mobile users in wireless data broadcast systems. We focus on two classical location-based queries[1]: window queries and k nearest neighbor searches. The former is to retrieve all the queried objects that fall inside the query window centered at a query point, and the latter is to retrieve the k objects located nearest to a query point. In both cases, the query point could be the current user location or on an anticipated user moving trajectory. In the design of DSI, we paid special attentions to the following three aspects: 1) the performance requirement with respect to energy conservation and access efficiency; 2) the inherently unreliable and error-prone wireless communication; and 3) the support of general location-based queries. To cater for client mobility, we consider not only the typical *snapshot* queries in which a query point is fixed, but also the *continuous* queries in which the query point moves along an anticipated movement trajectory. A continuous query is used in a situation where future client movement is projected. Taking the continuous nearest neighbor (CNN) query as an example, a client may want to know the nearest gas stations along an anticipated path from her current location to New York city. The returned result contains pairs of the nearest gas stations and their corresponding path segments (i.e., each pair consists of a gas station g and the segment where g is the nearest gas station). This information can be cached in a client and is particularly useful for the client to avoid continuously issuing queries while moving. Similarly, a continuous window query is to figure out pairs of qualified objects and their corresponding segments. The proposed DSI is an index structure that facilitates highly *efficient* and *reliable* access of location-based data in wireless data broadcast systems. It exhibits the following properties:

- It organizes data objects in a certain linear order (e.g., Hilbert Curve) that *naturally* fits the media of wireless data broadcast to facilitate efficient indexing and scheduling for broadcast.
- It has a fully distributed structure that allows query processing to start immediately and thus minimize the unnecessary waiting time for the arrival of the starting point of a search path. The property significantly shortens the access latency of the queried data.
- It provides multiple search paths for same objects and shares links on common search paths to minimize the bandwidth overhead of index information.
- It is very resilient to the inherent error-prone communication environment in wireless data broadcast. The clients can resume, instead of restarting, the query processing operation shortly after an error occurs.
- It efficiently supports a variety of location-based queries, including window query and k nearest neighbor (kNN) query, in both of the snapshot and continuous query modes.

The issues of developing efficient and error-resilient search algorithms for location-based queries are particularly challenging. The designs of traditional spatial indexes, e.g., R-Trees, are based on resident storage such as memory and disk which supports *backtracking* in tree traversal. Unfortunately, the linear property of wireless broadcast environments imposes a serious constraint of *sequential access* on R-trees. Thus, our search algorithms for DSI are based on the principle of following the linear access order but refining the search space and filtering out unqualified objects efficiently. Insights obtained from our analysis have led to a solution based on space refining and objects filtering and result in very efficient query processing algorithms.

To the best of our knowledge, this is the first work addressing location-based query processing issues in the error-prone wireless communication environment. It is observed that a node in traditional tree-structured indexes is only pointed by one parent node. As a result, the only access to a node will be lost if its parent/ascendant node could not be reached successfully. The search performance penalized by packet loss can be significant. Alternatively, DSI, by converging multiple search paths into one linear index structure, enables a fast resume of the search process. An extensive performance evaluation in both error-free and error-prone wireless communication environments shows that the proposed DSI performs significantly better than a variant of R-trees tailored for wireless data broadcast environments.

A preliminary version of this work has been reported in [21], where only the snapshot queries are considered. In this paper, for the sake of completeness, we extend the work to answer window queries and kNN queries in the more challenging continuous query mode. In addition, we provide an analytical model to study the performance of a primitive search algorithm, namely *Energy Efficient Forwarding (EEF)*, in both error-free and error-prone environments. The simulation is also extended to examine more performance aspects, including power consumption, computation cost and error resilience, on large datasets. Finally, we also discuss how to extend DSI to support queries on multiple data types.

The rest of this paper is organized as follows. A brief review of the wireless data broadcast, space filling curves, and existing work on air indexing for wireless data broadcast is provided in Section 2. The index

[1] Without causing any confusion, we use location-based queries and spatial queries interchangeably.

structure of DSI is detailed in Section 3, together with the description and detailed analysis of EEF, in both error-free and error-prone environments. Based on DSI, algorithms that support snapshot queries, as well as continuous queries, are presented in Section 4. Further, a simulation based performance evaluation is conducted in Section 5 to demonstrate the advantage of DSI in support of multiple spatial queries and its resilience under error-prone wireless communication environments. Finally, we conclude this paper in Section 6.

2 Preliminaries

In this section, we first describe the general system model, assumptions, and constraints of wireless data broadcast systems. Next, we discuss existing work of supporting spatial queries in wireless data broadcast environments.

2.1 System Model

Typically, a wireless data broadcast system consists of three parts: 1) a communication mechanism; 2) a broadcast server; and 3) a number of mobile clients. Figure 1 shows a high-level view of the system model. The communication mechanism consists of a number of wireless broadcast channels to disseminate data from the server to the clients. Without loss of generality, we assume only one broadcast channel is allocated in our study, without any uplink channel (since this is a push-based broadcast system). To answer queries, the mobile clients tune into the broadcast channel to retrieve data on air. Moreover, we assume clients can determine their current locations via some positioning technology, such as GPS. To simplify our study, we assume the entire service area, in which the clients can roam freely, is covered by one broadcast server (e.g., a satellite)[2].

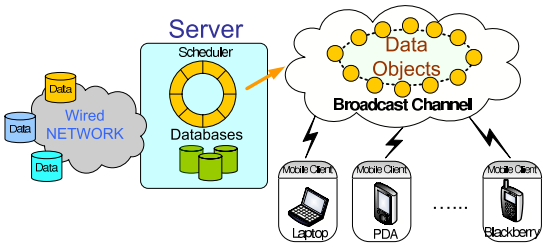


Fig. 1 A Wireless Data Dissemination System

The broadcast server is connected to data sources via high speed networks and thus can be considered as a logical data source for all the mobile users. An important task of the server is to determine and schedule the data objects for the broadcast program. A data object consists of a set of searchable attributes and a content body. Since our focus in this study is on location-based data access, we simply assume that the search attributes consist of geographical coordinates. A homogeneous data type (e.g. restaurants) is considered in this study. While complicated applications may likely issue complex queries that involve multiple types of data objects, our study represents an initial step towards this direction. In Section 3.4, we will discuss how to extend DSI to support multiple data types.

The server *periodically* disseminates data objects to its clients via the shared broadcast channel, and a complete broadcast of data objects is called a *broadcast cycle*. From the standpoint of mobile clients, data objects appear as a sequential stream along the time axis. A client may start retrieving data on air whenever it tunes into the broadcast channel. Thus, a broadcast cycle may logically start at any data object and end at the next appearance of the same data object. Finally, content updates of a data object are reflected between two successive broadcast cycles.

The performance of a wireless data broadcast system is measured by two criteria: *access efficiency* and *energy conservation*. As mobile clients are typically powered by batteries with limited capacity, the main concerns are 1) how fast a request could be satisfied; and 2) how battery energy of mobile clients could be conserved. To improve energy conservation, smart mobile devices can switch between two operation modes: *active mode* and *doze mode*. In doze mode, the clients suspend most energy consuming tasks like computation and communication whereas in active mode, some of those suspended tasks are performed. Energy conserving clients stay in doze mode most of the time and change to active mode occasionally when there is a need to perform energy consuming tasks. In the literature, two performance metrics, namely *access latency* and *tuning time*, are commonly used to measure access efficiency and energy consumption for mobile clients in a wireless data broadcast system, respectively [15,17,18][3]:

- Access Latency: The time elapsed from the moment a query is issued to the moment it is answered.
- Tuning Time: The time a mobile client stays in active mode to receive the requested data objects and index information.

2.2 Conventional Spatial Query Processing

In conventional spatial database research, many index structures have been proposed for the access of spatial

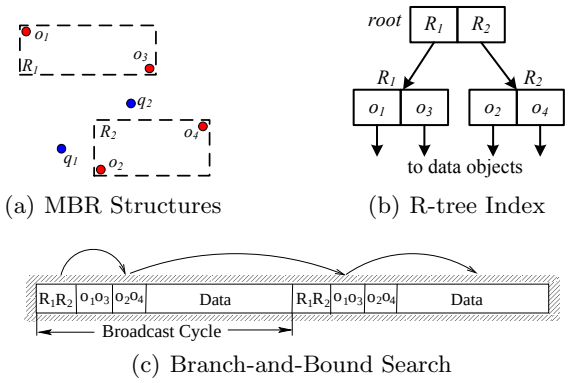
[2] The proposed indexing technique is also applicable to single cell in the cellular based mobile system. However, issues involving with the overlapping of services among the neighboring cells may need further investigation.

[3] In this study, while tuning time and access latency are used as the primary performance metrics to guide the design and optimization of DSI, we also investigate the power consumption and computational cost of mobile clients in the performance evaluation.

data, including R-tree [14], KD-tree [27], Quad-tree [28], etc. Among them, R-tree is most well received for its simplicity, efficiency, and compatibility to handle a wide range of spatial data and queries. The basic idea is to approximate a spatial object with a minimal bounding rectangle (MBR) and to index these MBRs recursively using larger MBRs. Each node in the index tree contains a number of entries. Each entry in an internal node contains a child-pointer to a lower level node in the tree and an MBR covering all the rectangles in the lower nodes in the subtree. In a leaf node, an entry consists of a pointer to the data object and an MBR which bounds the data object's spatial region. Variants of the R-tree differ from each other in terms of the criteria used to construct the index. Figure 2(b) shows the structure of an R-tree, where corresponding objects and MBRs are depicted in Figure 2(a).

R-tree based search algorithms typically follow a *branch-and-bound* approach to dynamically adjust the search space based on the positions of the query point and objects. For example, suppose that there are two query points, q_1 and q_2, as shown in Figure 2(a). For query point q_1, after accessing the root, it visits R_2 first since it is closer to R_2 than to R_1. In R_2, the NN of q_1 is o_2. Hence, it records the current minimal distance, $|q_1, o_2|$[4]. Because $|q_1, o_2|$ is shorter than the minimum distance from q_1 to R_1, the search is completed. Similarly, for query point q_2, it first examines the root and then R_2. Next, it examines R_1 since the current minimal distance, $|q_2, o_4|$, is longer than the minimum distance from q_2 to R_1. As illustrated, NN search for R-tree dynamically traverses the MBRs according to the given query point, which introduces *backtracking*, i.e., the search goes back to its parent node and switches to other branches more likely to contain the result objects. This operation is well supported in random-access media such as the main memory and hard disks.

In a wireless broadcast channel, however, data objects are broadcast based on a pre-defined sequence (called a *broadcast program*) and thus a data object is only available when it is on air. Consequently, search algorithms designed for random access storages may incur a significant access latency. For example, Figure 2(c) illustrates branch-and-bound search of the R-tree depicted in Figure 2(b) on air. Assuming that a search algorithm chooses to visit the node R_2 and then node R_1, after visiting the root node, yet the index nodes are broadcast in the order of root, R_1, and R_2. Consequently, if a client intends to backtrack to R_1 after visiting node R_2, it has to wait until the next cycle because R_1 has already been broadcast in the current cycle. This occurs every time when a search order differs from the broadcast order and thus significantly extends the access latency. Alternatively, the client can perform the search in accordance with the broadcast order, i.e., visiting root, R_1, and then R_2, in order to fit the sequential access property of wireless data broadcast. However, this conservative approach may retrieve some unnecessary objects avoidable based on branch-and-bound search algorithm and hence results in a longer tuning time and more energy consumption.



(a) MBR Structures (b) R-tree Index

(c) Branch-and-Bound Search

Fig. 2 Linear Access on Broadcast Channel

From this example, we can easily observe the deficiencies of R-tree index in wireless data broadcast systems. Besides, due to the unique characteristics of wireless data broadcast, design requirements for air indexes are distinct from those for the traditional disk-based indexes:

- The size of air index has a direct impact on the access latency and thus needs to be small.
- Air index can only be accessed sequentially. Thus, the index structure and search algorithms should take the property of sequential access into account.
- Query processing based on wireless data broadcast should start as soon as possible instead of waiting for a specific index node (e.g., the root of an index tree) to arrive.
- Additionally, query processing based on wireless data broadcast should be resilient to the error-prone wireless communication environment. In the events of error while receiving data, the client needs to be able to resume (instead of *restarting*) the search shortly.

2.3 Air Indexing

Without any auxiliary information about the broadcast schedule of data objects, a client is forced to exhaustively listen to the wireless channel for requested objects. For instance, to locate an object closest to its current position, a client has to download the whole data set. This clearly consumes a lot of energy since the client has to stay in *active* mode for one entire broadcast cycle.

Air indexing techniques are often used for conserving the energy of mobile clients [18,20]. The basic idea is that the broadcast server pre-computes indexing information (including searchable attributes and delivery time of data objects) and interleaves it with data objects on the broadcast channel. Through the air index, mobile

[4] Function $|a, b|$ returns the Euclidean distance between two points a and b.

clients are aware of the arrival time of desired data objects by examining the index information. Long before the arrival of the objects, the clients can switch to doze mode and only wake up to active mode when the objects are about to be broadcast. Similarly, appending to each data object, the delivery time of the next index helps the clients to schedule the sleep time for subsequential data access. Consequently, the search of data objects is facilitated.

Imielinski et al. has extended B^+-trees to assist the access of broadcast data. Two approaches, namely, (1, m) and *distributed index*, are proposed to interleave the index and data in a broadcast channel [18]. The former treats the whole index as one segment and replicates the segment m times within one broadcast cycle. An index scan is guaranteed to be finished by visiting one index segment. However, the clients suffer from a longer access latency since m identical index segments extend the overall broadcast cycle significantly. The distributed index replicates only the top part of an index tree (see Figure 3 for illustration), and the replicated part is only a small portion of the entire index. It has been shown that the distributed index scheme is more efficient than (1, m) in terms of access latency.

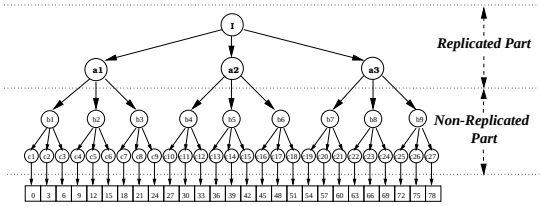


Fig. 3 A Distributed Index for Broadcast

Chen et al. proposed a broadcast index based on unbalanced tree tailored for non-uniform data access [6]. By organizing the data objects with low access frequencies in low-level nodes of the index and putting frequently accessed data in high-level nodes, the search cost of frequently accessed data is minimized. Signature and hashing methods have also been developed for data broadcast systems to support equality queries [15,20]. Signatures and hash index frames are interleaved with data objects or a group of data objects. By matching the hash value or signature of a query with that of indexed attribute values in the data objects, a client can easily avoid retrieving unwanted data in the broadcast channel.

Ideas of indexing the attribute ranges for exponentially increasing number of data objects were discussed in different contexts, e.g., *Chord* [33], *flexible index* [17] and *exponential index* [37]. 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 the tuning time in support of simple searches of broadcast data based on a *single* attribute. None of them considered complex location-based queries, as we address in this paper.

2.4 Processing of Location-Based Queries

In the literature, different approaches have been proposed to facilitate the processing of location-based queries (LBQs). Dunham and Kumar have done lots of pioneer studies in this area. They presented a query formalization which considers both location-dependent data and location-independent data in [5], proposed an architecture for the processing of LBQ in [30], and developed semantic caching for managing location-dependent data in mobile environments in [26]. The placement of location-dependent data along wireless broadcast channels has been addressed in [39,40], and the privacy issue of accessing location-dependent data has been addressed in [4, 11,24]. In this paper, we focus on supporting location-based queries, namely, window queries and kNN queries, in error-prone wireless broadcast systems.

Several air indexes have been recently proposed to support broadcast of spatial data (i.e., location dependent data) [38,42,43]. These studies can be classified into two categories, according to the nature of the queries supported. The first category focuses on retrieving data associated with some specified geographical range, such as "Starbucks Coffee in New York City's Times Square" and "Gas stations along Highway 515". A representative is the index structure designed for *DAYS* project [2, 10]. It proposes a location hierarchy and associates data with locations. The index structure is designed to support queries on various types of data with different location granularities. The authors exploit an important property of the locations, i.e., containment relationship among the objects, to determine the relative location of an object with respect to its parent that contains the object. The containment relationship limits the search range of available data and thus facilitates efficient processing of the supported queries. In brief, a broadcast cycle consists of several sub-cycles, with each containing data belonging to the same type. A major index (one type of index buckets) is placed at the beginning of each sub-cycle. It provides information related to the types of broadcast data, and enables clients to quickly jump into the right sub-cycle which contains her desired data. Inside a sub-cycle, minor indexes (another type of index buckets) are interleaved with data buckets. Each minor index contains multiple pointers to the data buckets with different locations. Consequently, a search for a data object involves accessing a major index and several minor indexes.

The second category focuses on retrieving data according to specified distance metrics. An example is nearest neighbor search based on Euclidian distance. *D-tree* is a paged binary search tree to index a given solution space in support of planar point queries. *Grid-partition*

index is specialized for the (single) Nearest-Neighbor problem [43]. *Hilbert Curve Index (HCI)* is designed to support general spatial queries, including window queries and nearest-neighbor queries. It adopts a B^+-tree to index broadcast data objects according to the Hilbert Curve order. Conventional spatial index R-tree has also been adapted to support kNN search in broadcast environments [12].

The problems of those two categories are different in nature and hence the index structure designed for one category cannot be directly applied to the other. DSI falls into the second category. Although DSI employs the Hilbert curve values to order spatial objects as HCI, the index structures and search algorithms are completely different.

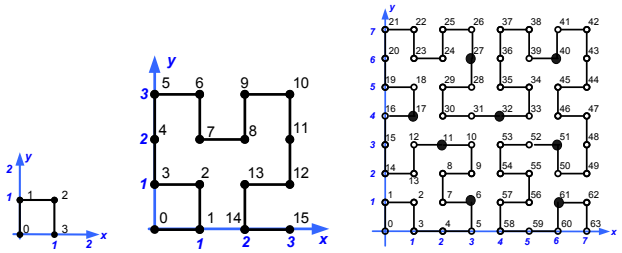
2.5 Hilbert Curve

Since most spatial queries search for objects that are closely located, one strategy for wireless data broadcast is to schedule (spatially) near objects close to each other in the broadcast program. However, spatial objects reside in a two- or three- dimensional space whereas data objects on air are broadcast in a linear media. To keep neighboring objects in a high dimensional space remain close to each other on the broadcast channel, a space-filling curve (e.g., *Hilbert Curves* (HC) [13]) can be adopted to schedule broadcast of data objects based on the coordinates on the curve. As such, the data locality on the broadcast channel can meet the access locality required by spatial queries. DSI adopts this idea to build an air index upon space filling curve[5].

Figure 4(a) shows the basic Hilbert curve of order 1. To derive a curve of order i, each vertex of the basic curve is replaced by a curve of order $(i-1)$, which may be strategically rotated and/or reflected to fit the new curve. The Hilbert curves of orders 2 and 3 are depicted in Figure 4(b) and Figure 4(c), respectively. The numbers, called *HC values* in this paper, represent the visiting orders of different points in the Hilbert curve. For example, curve H_2 illustrates a (4×4) grid, where the point located in $(x = 1, y = 1)$ has the HC value 2.

Given the HC and its coordinate mapping function, it is easy for a client to convert between coordinates and HC values. Let n be the number of bits assigned to represent a coordinate in one dimension, and d be the dimensionality of the original search space, the expected time complexity for the conversion is $O(n^d)$. Since n is a constant and d is usually 2 or 3, the conversion overhead is fixed. The detailed conversion algorithm is available in [25].

The HC provides a linear order of data objects for broadcast, while maintaining maximal locality amongst nearby data objects. The remaining challenge to be addressed is how to design an air index such that spatial queries can be efficiently answered. It is the focus of this paper.



(a) H_1 (b) H_2 (c) H_3

Fig. 4 Hilbert Curve of Order 1, 2, and 3

3 Distributed Spatial Index (DSI)

In conventional spatial databases, spatial queries are well supported by tree-based indexes, such as R-tree and its variants. Based on searching criteria, searches start from the root node of a tree through a sequence of intermediate nodes to reach leaf nodes pointing to target spatial objects. However, employing a tree-based index on a linear broadcast channel to support spatial queries results in several deficiencies. Firstly, clients can only start the search when they retrieve the root node in the channel. Replicating the index tree in multiple places in the broadcast channel provides multiple search starting points, shortening the initial root-probing time. However, a prolonged broadcast cycle leads to a long access latency experienced by the clients. Secondly, a wireless broadcast media is not error-free. In case of losing intermediate nodes during the search process, the clients are forced to either restart the search upon an upcoming root node or scan the subsequential broadcast for other possible nodes in order to resume the search, thus extending the tuning time. Thirdly, backtracking, commonly used in traversing tree-based index, is no longer efficient in linear access medium. As a result, tree-based indexes are not appropriate for supporting spatial queries in data broadcast environments.

Motivated by the need of an index structure supporting spatial queries in broadcast environments, we present a fully distributed spatial index structure, namely *DSI*, in this section. Very different from tree-based indexes, DSI is not a hierarchical structure. Index information of spatial objects is fully distributed in DSI, instead of simply replicated, in the broadcast. With DSI, the clients do not need to wait for a root node to start the search. The search process launches immediately after a client tunes into the broadcast channel and hence the initial probe time for the index information is minimized. Furthermore, in the event of data loss, clients resume the

[5] Please note that space filling curves other than HC can also be used for DSI.

search quickly. In this section, we focus on a basic operation upon DSI, i.e., EEF algorithm, and discuss the possible approaches to extend DSI to support various types of data objects. The detailed search algorithms for other types of spatial queries will be discussed in the next section.

3.1 The Index Structure

Hilbert curve (HC) is adopted in DSI to determine broadcast order of data objects. Data objects of the same storage size, mapped to point locations in a 2-D space, are broadcast in the ascending order of their HC values. Suppose there are N objects in total, DSI chunks them into n_F *frames*, with each having n_o objects ($n_F = \lceil N/n_o \rceil$). We use the space covered by Hilbert Curve shown in Figure 4(c) as a running example, with solid dots representing the locations of data objects (i.e., $N = 8$). Figure 5 demonstrates a DSI structure with n_o set to 1, i.e., each frame contains only one object. Notation o_{hc} represents an object o having hc as its HC value. For example, o_6 represents the object spatially located in $(x = 3, y = 1)$, as shown in Figure 4(c).

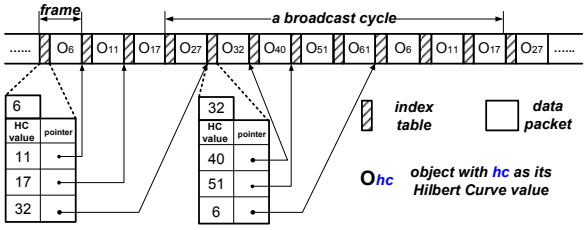


Fig. 5 DSI for the Running Example

In addition to objects, each frame has an index table as its header, which maintains information regarding to the HC values of the objects included in the current frame as well as information about data objects to be broadcast after specific waiting interval[6]. Every index table keeps n_i entries, each of which, τ_j, is expressed in the form of $\langle HC'_j, P_j \rangle$, $j \in [0, n_i)$. P_j is a temporal pointer to the r^j-th frame after the current frame, where r (> 1) is an exponential base (i.e., a system parameter), and HC'_j is the HC value of the first object inside the frame pointed by P_j. When n_i is set to 1, each index table contains one entry pointing to the next frame. In this case, DSI becomes a linked list of frames. When n_i is set to $\lfloor log_r(n_F) \rfloor$, each index contains pointers to the objects in the next frame, the r^1th frame, the r^2th frame, ..., and the frame that is at least $1/r$ of broadcast cycle away from the current one. Logically, a broadcast cycle starts from any frame F, and ends right before the next appearance of frame F. When n_i is larger than $\lfloor log_r(n_F) \rfloor$, pointers in the index table of frame F may refer to the frames that are at least one broadcast cycle away from the current frame F. In other words, the index table of F may contain pointers that span more than one broadcast cycle. Since a broadcast cycle already contains the entire data set, we set $\lfloor log_r(n_F) \rfloor$ as the value of n_i. In addition to τ_j, an index table keeps the HC values HC_k ($k \in [1, n_o]$) of all the objects obj_k that are contained in the current frame. This index information, although occupying little extra bandwidth, can provide a more precise image of all the objects inside current frame. During the retrieval, a client can compare HC_k's of the objects against the one she has interest in, so the retrieval of any unnecessary object (whose size is much larger than an HC value) can be avoided.

Let us go back to the example shown in Figure 5. Suppose $r = 2$ and $n_o = 1$, each index table has $log_r(n_F) = 3$ entries since $n_F = 8$. The index tables corresponding to frames of data objects o_6 and o_{32} are shown in the figure. Take the index table for frame o_6 as an example: τ_0 contains a pointer to the next upcoming (2^0-th) frame whose first object's HC value is 11, τ_1 contains a pointer to the second (2^1-th) frame with HC value for the first object being 17, and the last entry τ_2 points to the fourth (2^2-th) frame. It also keeps the HC value 6 of the object o_6 in the current frame.

3.2 Energy Efficient Forwarding

Since DSI distributes index tables along with all the frames in the broadcast, lookup of interested objects with known spatial coordinates is facilitated by traversing one or multiple frames. In this section, we discuss energy efficient forwarding (EEF), the most basic search algorithm for DSI. As the name indicates, this algorithm, good for saving energy, has a small cost in terms of tuning time. The EEF algorithm is designed to also accommodate the situation where the requested object with a specified coordinate may not exist.

The pseudo-code of EEF is depicted in Algorithm 1. Here, we use an example to explain the search algorithm. Looking for an object O spatially located in point $p = (x, y)$, a client first translates the coordinate p to the corresponding HC value HC_p and starts tuning into the broadcast channel for the object. After synchronizing the broadcast channel, the client retrieves the index table of its first frame. It compares HC_p with HC_k to check whether the object O is contained in the current frame. If a match is found, the object is retrieved and the search is terminated. If HC_p is within the range bounded by HC_1 and HC_{n_o}, but there is no match, the search is terminated with an empty result. Otherwise, the client needs compare the HC'_i maintained by the index table, and follows the pointer P_i with $HC_p \in [HC'_i, HC'_{(i+1) \bmod n_i})$[7].

[6] This waiting interval can be denoted as a *temporal pointer* represented by the number of data frames apart from the current frame.

[7] If HC'_i is larger than $HC'_{(i+1) \bmod n_i}$, $[HC'_i, HC'_{(i+1) \bmod n_i})$ is replaced by the ranges $[HC'_i, HC_{max}) \cup [0, HC'_{(i+1) \bmod n_i})$.

Algorithm 1 EEF

Input: The HC value HC_p of the spatial p locating at (x, y) in a 2-D space;
Output: Requested object;
Procedure:

```
1: begin the initial probe and retrieve the first frame F;
2: while (F is not empty) do
3:    retrieve the index table associated with F;
4:    if (HC_1 ≤ HC_P ≤ HC_{n_o}) then
5:       /*check objects of the current frame*/
6:       for (k := 1; k <= n_o; k := k + 1) do
7:          if (HC_P = HC_k) then
8:             retrieve the kth object obj_k in the frame;
9:             return obj_k;
10:      no match is found, return ∅;
11:   for (P := NULL, i := 0; i < n_i; i := i + 1) do
12:      /* check the index entry τ_i */
13:      if (HC'_i ≤ HC_p < HC'_{(i+1) mod n_i}) then
14:         P := P_i; break;
15:      else if (HC'_{(i+1) mod n_i} < HC'_i) and (HC'_i ≤ HC_p or
         HC_p < HC'_{(i+1) mod n_i}) then
16:         P := P_i; break;
17:   if (P ≠ NULL) then
18:      switch to doze mode until frame F pointed by P
         arrives;
19:   else
20:      F := NULL;
```

EEF is a basic operation and a building block for search algorithms of other more complex queries. In the following, we mathematically analyze its performance, in terms of tuning time and access latency. The notations used in our analysis are summarized in Table 1.

Notation	Description
N	the number of data objects.
C	the capacity of a packet.
r	the exponential base ($r > 1$).
s_o	the size of a data object.
n_o	the object factor (i.e., the number of objects in a frame).
n_F	the number of frames within one broadcast cycle, $n_F = \lceil N/n_o \rceil$.
s_{HC}	the size of a Hilbert Curve value.
s_p	the size of a pointer.
s_t	the size of a index entry, $s_t = s_{HC} + s_p$.
n_i	the number of index entries in one index table, $n_i = \lfloor log_r(n_F) \rfloor$.
s_i	the size of an index table, $s_i = n_i \cdot s_t$.
s_F	the size of a frame, $s_F = s_i + n_o \cdot (s_{HC} + s_o)$.

Table 1 Definition of Notations

The tuning time of EEF is the cumulated time when clients stay in active mode to retrieve requested objects and/or useful index information pointing to requested objects both directly or indirectly to facilitate EEF on DSI. Suppose a client tunes into the broadcast channel and the first frame received is F_m, while the requested object is located in frame F_{m+l}, which is l ($0 \leq l \leq n_F - 1$) frames behind the current frame. The tuning time to reach the target frame F_{m+l} (from F_m) is denoted by $t(l)$, which is dependent on distance l and the exponential base r. Suppose that the client can start the search at any time. The lag time of the requested object and the search starting time is uniformly distributed in between 0 and $n_F - 1$ frames. Therefore, the expected tuning time in terms of number of frame accesses is expressed in Equation (1). The first term, $\frac{C}{2s_F}$, represents the initial probe for the first complete frame. As each packet contains a pointer to the next frame, the initial probe on average requires retrieval of half of a packet. The second term is the average number of frames accessed. $t(l)$ is expressed as a recursive form as stated in Equation (2). Here, x is the maximum value from $\{r^0, r^1, r^2, \cdots, \lfloor n_F/r \rfloor\}$, that does not exceed l.

$$E(tuning) = \frac{C}{2s_F} + \frac{1}{n_F} \sum_{l=0}^{n_F-1} t(l) \quad (1)$$

$$t(l) = \begin{cases} 1, & l = 0; \\ t(l-x) + \frac{s_i}{s_F}, & l > 0. \end{cases} \quad (2)$$

The exponential base, r, can be used to control the number of index entries in DSI table. Obviously, with more entries indexing data objects in a broadcast cycle, tuning time can be better improved. In the following, we set $r = 2$ in order to analyze the access efficiency of DSI. For clarity of presentation, we set $s = s_o + s_{HC}$. Thus, the average latency for energy efficiently reaching a frame is derived in Equation (3).

$$E(latency) = \frac{s_F}{2} + \frac{s_F \cdot n_F}{2} = \frac{s_i + n_o \cdot s}{2} \cdot (1 + \frac{N}{n_o}) \quad (3)$$

3.3 Error Resilience

The above description is based on an error-free wireless communication environment, in which there is no signal interference or packet loss. In practice, the wireless environment is inherently unreliable and error prone due to radio propagation attenuation, fading and noises. In such an environment, link errors occur frequently and thus an error-resilient air index that can quickly resume interrupted query processing is highly desirable. DSI naturally is an excellent scheme to provide this function owing to its fully distributed index structure. If a frame is lost, the query processing can continue in the next frame based on alternative search paths. Thus, performance deterioration is minimized. This is a great advantage of DSI over other air indexes.

For a conventional tree-based air index, any node is only pointed by one parent. Therefore, the immediate

In this paper, HC_{max} represents the maximal HC value within the search space

access to a node is lost if its parent/ascendant node is not reached successfully. The client has to either wait for rebroadcast of the lost node or blindly scan all the following packets from the wireless channel until the desired node is received. However, both approaches are not efficient. The former incurs an extremely large access latency, while the latter wastes scarce power resources by scanning lots of useless nodes.

Differently, DSI distributes the index information over the whole broadcast cycle. From any given starting frame, there is always a path available to reach a target frame efficiently and thus enables any interrupted search to resume immediately. Take Figure 8 as an example, which sets frame F_8 containing object o_{61} as the target frame. Table 2 lists all the search paths resumed at different frames.

Starting Frame	Search Path
F_1	$F_1 -> F_5 -> F_7 -> F_8$
F_2	$F_2 -> F_6 -> F_8$
F_3	$F_3 -> F_7 -> F_8$
F_4	$F_4 -> F_8$
F_5	$F_5 -> F_7 -> F_8$
F_6	$F_6 -> F_8$
F_7	$F_7 -> F_8$

Table 2 Search Paths Under DSI

To facilitate our study on error-prone wireless environments, a variable θ is introduced to control the probability of a frame loss. When θ is 0, no frame is lost which is the ideal environment considered in the previous subsection. On the other extreme, θ is 1 and all frames are lost. In the following, we examine the resilience of DSI under various θ's via theoretical analysis. The results of experimental simulation will be presented in Section 5.

Recall that Equation (2) presents the required tuning time of accessing a frame which is l frames away from the current frame under the ideal environment. If the current frame cannot be received successfully, the client will not obtain the index information provided by that frame. Therefore, the search will be resumed by listening to the next frame. Taking the frame loss into consideration, we derive the tuning time performance in presence of broadcast errors in Equation (5).

$$E(tuning, \theta) = \frac{C}{2s_F} + \frac{1}{n_F} \sum_{l=0}^{n_F-1} t(l, \theta) \quad (4)$$

$$t(l, \theta) = \begin{cases} 1, & l = 0; \\ \big(1-\theta\big) \cdot \big(t(l-x, \theta) + \frac{s_i}{s_F}\big) + & \\ \theta \cdot \big(t(l-1, \theta) + \frac{s_i}{s_F}\big), & l > 0. \end{cases} \quad (5)$$

Here we only consider the situation where a small number of frames ($\leq n_F$) are lost. In case that a long disconnection occurs (e.g., more than n_F frames are lost), the pointer information obtained from previous probing is no longer valid. Consequently, the search has to restart anyway. Similarly, if the target frame which contains the needed objects is lost, the client has to wait for the next cycle to access the desired data, no matter which kind of indexing technique is employed. Therefore, without loss of generality, we assume that the target frame will be received successfully in this analysis, i.e., $t(0, \theta) = 1$. Since the target frame can be reached from any frame, the access latency will not be affected by the link errors. Therefore, the average access latency of DSI is still a half of the broadcast cycle, as shown in Equation (3).

3.4 Multiple Data Types

In the above discussion, we assume all the data objects are of the same data type (e.g., restaurants). In many real life applications, users of diverse interests might issue queries on objects of different types. For examples, visitors shopping at the Fifth Avenue might look for nearby ATM when they run out of cash, while others might want to find a nearby restaurant for lunch. In this section, we briefly discuss two approaches to extend DSI for supporting queries of different data types, namely the *sequential* approach and the *integrated* approach.

Sequential Approach. The sequential approach uses a single channel to broadcast data objects of different types. It splits the broadcast cycle into sub-cycles with each sub-cycle corresponding to a single data type. The index table of each frame contains P_j to facilitate the search within a sub-cycle of a particular data type T_i. To direct searches of different data types to the right sub-cycle, the index table also stores $\langle T_i, P_i \rangle$, where T_i is the unique identifier of a particular data type and P_i is the pointer to the start of the corresponding sub-cycle.

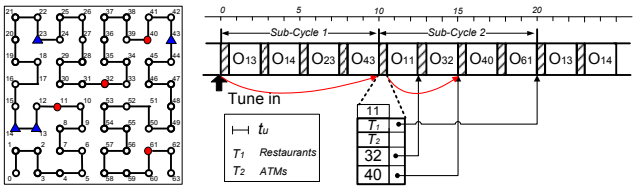


Fig. 6 A Multiple Types Dataset and the Sequential Broadcast Approach

Figure 6 depicts an example in which two types of data objects are supported (i.e., restaurants and ATMs), denoted by filled circles and filled triangles, respectively. A broadcast program is organized in accordance with the sequential approach. Suppose a user issues a query q to retrieve the ATM located in the position ($x = 6$, $y = 6$, i.e., its HC value is 40). The user tunes into the broadcast channel to retrieve an index table (the first index table as illustrated in this example). Thereafter, it switches to doze mode until the sub-cycle corresponding

to the desired data type (i.e., sub-cycle 2) arrives. The user downloads the first index table in the sub-cycle and follows the second link to access data item O_{40}. Assume it takes 0.5 time unit t_u to broadcast an index table and 2 time units to broadcast a frame which contains one single data item. The access latency for q is 17.5 time units and the tuning time is 3.5 time units.

Integrated Approach. The sequential approach supports searches on particular data types. However, a user may issue a query that involves multiple data types. For example, a user is going to attend his friend's birthday party in a restaurant. He wants to visit a nearby gas station, a bakery and a flower shop on his way to the restaurant. He can tune into three different sub-cycles in the sequential approach to retrieve the targeted gas station, bakery, and flower shop. As those targeted objects might be located closely, it is a waste to conduct spatial queries (e.g., CNN search in this example) on the same area multiple times. Motivated by this observation, we propose the integrated approach, which aims at using one spatial index to support queries on different types of objects.

The integrated approach broadcasts all the data objects, regardless of their data types, in one wireless channel in the order of their HC values. A broadcast program organized in accordance with the integrated approach for the same sample datasets defined in Figure 6 is depicted in Figure 7. The original index table keeps $\tau_i = \langle HC'_i, P_i \rangle$ tuples which guide the search to move towards the targeted items via EEF. However, as data objects of different types are integrated in one broadcast program, a bit-string tag is appended to each τ_i to indicate the types of the data objects with HC values ranging between HC'_i and $HC'_{(i+1) \bmod n_i}$. In other words, the pointer P_i serves as an entrance to the data stored in the next 2^i-th to next 2^{i+1}-th frames, and the bit-string tags the data types of the data objects contained by those frames. The length of the corresponding bit-string is bounded by the number of data types. In our example, two data types, restaurants and ATMs, are tagged by a 2-bit string where the first bit corresponds to restaurants and the second bit corresponds to ATMs. As shown in Figure 7, the index table of item O_{32} has three tuples. The first one points to the next frame, i.e., F_6 containing the object O_{40}. Since O_{40} is the only object contained in the next frame, it must be an ATM as its bit string is 10. The second one points to the 2^1-th frame and the bit-string 11 indicates that frames F_7 and F_8 contain objects from both types. Similarly, the third tuple points to the 2^2-th frame. The bit-string again is 11 which indicates that frames F_9, F_{10}, F_{11}, and F_{12} contain objects of both types. When a user issues a query, it only evaluates those frames containing the desired data types. When data objects of multiple types share similar geographical distributions, the integrated approach can be employed to avoid redundant index information.

How to organize the broadcast channel to facilitate access of data objects of different data types is a topic which deserves further investigation. Due to the space limitation, we discuss the above two possible approaches that can support queries of heterogeneous data types. Since the performance of these approaches is dependent on multiple factors, such as the distribution of data objects and the query patterns of different users, we leave the issue of tuning the performance in real applications and the development of other approaches as a future study to work on.

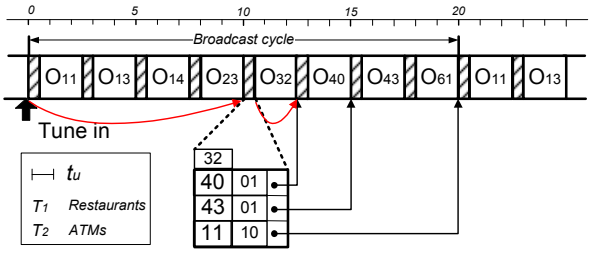


Fig. 7 The Integrated Broadcast Approach

4 Spatial Queries Processing

In this section, we first discuss the search algorithms based on DSI for snapshot window queries and nearest neighbor (NN) searches, two classical spatial queries. Considering the scenario where mobile clients may issue queries while moving, we also discuss the support of continuous window queries and continuous NN searches on DSI. Finally, we discuss briefly how to extend all the proposed search algorithms in a homogeneous data type scenario.

4.1 Snapshot Query Processing On Air

Snapshot spatial queries are issued from static query points, i.e., the current locations of mobile clients. Example queries include window queries and k-nearest neighbor (kNN) queries (where $k \geq 1$). A window query retrieves spatial objects inside a query window, as "locate all the pubs within this town". kNN query retrieves k objects that are closest to a given query point among all the objects. An example of kNN query is "locate three Chinese restaurants nearest to my current position". We discuss the search algorithms in the following two subsections.

4.1.1 Window Queries Query windows can be of different shapes. Without loss of generality, we assume a rectangular query window specified by two ranges in respective dimensions. Therefore, given a window query Q, the corresponding window W_Q is specified as $[x_1, x_2] \times$

$[y_1, y_2]$. The search needs to find objects with their coordinates $p \in [x_1, x_2] \times [y_1, y_2]$. Since Hilbert Curve is a space-filling curve, the intersections between the curve and the boundary of the query window partition the curve into several segments, each of which corresponds to a HC value subsequence. For any segment, it is either fully inside the query window, or outside the window. In other words, W_Q is transformed to a set of HC value subsequences (called *targeted segments*) whose segments are inside W_Q. The basic idea of processing window queries is to employ the EEF operation to go from a targeted segment to the next targeted segment energy-efficiently.

Since an HC is linear, a segment seg within W_Q must share some common endpoints with segments outside W_Q. All these common endpoints must be on the boundary of W_Q. Hence, the window query algorithm first detects all the intersections between the HC and the boundary of W_Q. Without loss of generality, we assume that all the segments located inside the query window form a targeted segments set H. Upon receiving an index table, the client sequentially scans each entry and follows the first point P_i with the range $\{HC'_i - HC'_{i+1}\}$ overlapping with some segment seg_j of H. The qualified objects contained in the frame pointed by P_i are thereafter retrieved and the segment seg_j is refined by removing the overlapped part. This process continues until H becomes empty. The detailed pseudo-code of the window query algorithm is provided in Algorithm 2.

Algorithm 2 Window Query

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

1: compute the targeted segments set H, with each segment seg_i denoted by the range $\{H_{2i-1}, H_{2i}\}$;
2: begin the initial probe and find the pointer P to the first frame F_s; $R := \varnothing$;
3: **while** H is not empty **do**
4: switch to doze mode until frame F_s pointed at by P arrives;
5: **for** $(i := 1; i <= n_o; i := i + 1)$ **do**
6: **if** (HC_i is bounded by some targeted segment seg_j) **then**
7: $R := R \cup \{obj_i\}$; refine segment seg_j;
8: **for** $(P := NULL,\ i := 0; i < n_i; i := i + 1)$ **do**
9: refine targeted segments based on HC'_i;
10: **if** $(HC'_i < HC'_{(i+1) \bmod n_i})$ and $(\{HC'_i - HC'_{(i+1) \bmod n_i}\}$ overlaps some targeted segments) **then**
11: $P := P_i$; *break*;
12: **else if** $(HC'_{(i+1) \bmod n_i} < HC'_i)$ and $(\{HC'_i - HC_{max}\}$ or $\{0 - HC'_{(i+1) \bmod n_i}\}$ overlaps some targeted segments) **then**
13: $P := P_i$; *break*;
14: return R;

Figure 8 illustrates the window query processing with DSI. Suppose the shaded area in the figure is a query window. The client first detects all the targeted segments. In this example, the set H has three targeted segments, $\{10-11\}$, $\{28-35\}$, and $\{52-53\}$, all highlighted by dashed lines. 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 targeted segment $\{10-11\}$ from H. By combining index tables in F_1 and F_2, the client has more knowledge of the objects, i.e., $(6, 11, 17, 27, 32, 40, \cdots)$. Thus, it 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} are found not to exist. As Figure 8 where the client skips the downloading of dark frames shows, the access latency is 8 frames and the tuning time is 5 frames.

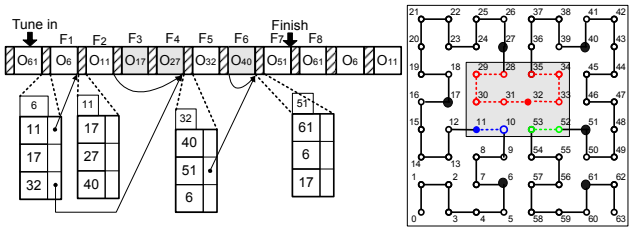


Fig. 8 Window Query Processing

4.1.2 K Nearest Neighbor Queries To determine whether an object o is qualified for a kNN search, a search algorithm needs to know not only locations of the object o and the query point, but also the locations of other candidate objects. However, algorithms based on a distributed index environment (e.g., DSI) have no complete knowledge of the objects distribution. As a result, it is impossible for a client to determine a *precise* search space which contains only those k qualified objects at the time the query is issued. The basic idea behind the kNN search algorithms for DSI is to determine a search space based on what clients know and then dynamically shrink the space as more index information is received.

The initial search space for a kNN query is *the entire search space*, which covers all the data objects, and the corresponding targeted segment is $\{0, HC_{max}\}$. As a client tunes into the channel and receives index tables, the existence of some objects is confirmed. The search space can be shrunk to a circle centered at the query point and containing k nearest objects out of those objects known so far. Correspondingly, the targeted segment is refined by removing HC values of both visited objects and those located outside the search space. The refining process continues and the retrieval of objects happens when objects are within the current search space.

Finally, the search completes when all k NN objects are retrieved and the targeted segment set becomes empty.

Obviously, how fast a search space can be shrunk from the entire search space to the final precise one directly affects the search performance. If the query point is located far away from the objects contained in the current broadcast frame, the initial circle could be very large because the index table has very limited information about the distribution of data objects close to the query point (due to exponential increase of data objects covered by index table entries). Consequently, many candidates will fall inside the large search circle, and the client has to download more objects and retrieve more index tables than necessary to refine the search radius. On the other hand, if a query point is close to the objects contained in the current broadcast frame, the search space will converge very rapidly and the search process typically will terminate quickly, because there are more information about data objects around the query point. In other words, clients can either follow the broadcast order to refine the search space gradually or jump to the frame closest to the query point for a quicker refinement. According to how to refine the search space, two strategies for kNN query processing, namely *conservative* and *aggressive*, are proposed.

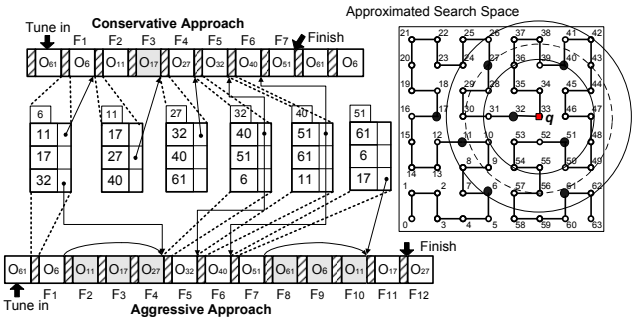


Fig. 9 kNN Query Processing

Conservative Approach.
The conservative approach does not skip any data object that is possible to be included 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 guarantees a kNN search to be completed within one broadcast cycle and hence has small access latency. However, clients might suffer from a relatively higher energy expense due to the slow search space convergence which turns out to trigger more objects retrieval. The algorithm is depicted in Algorithm 3.

For better illustration, we use an example (as shown in Figure 9) to explain the process. Suppose a client located at the spot q labelled by HC value 33 issues a 3NN query. Initially the search space covers an entire spatial region and the targeted segment is $\{0-63\}$, assuming the client knows the total search space is a 8×8 grid. The client tunes into the channel as depicted, and the first frame it receives is F_1. According to the index table of F_1, the client knows the existence of objects o_6, o_{11}, o_{17}, and o_{32}. Tentatively, it determines that objects o_6, o_{11}, and o_{32} are the three nearest neighbors known so far. Thus, the search space is refined to the solid-lined circle crossing o_6 (see Figure 9). The targeted segments are revised to $\{6-11\}$, $\{24-57\}$, and $\{61-62\}$ accordingly by removing those outside the search circle.

The client downloads object o_6 and follows the first pointer P_0 to access frame F_2. The existence of objects o_{27} and o_{40} is therefore detected. Objects o_{27}, o_{32}, and o_{40} become the new 3NN candidates, and the search space is further reduced to the dashed circle across o_{27}. Consequently, segment $\{6-11\}$ is removed and the targeted segments are refined to $\{26-28\}$ and $\{31-56\}$. The client skips object o_{11} and follows the second pointer P_1 to access frame F_4 since it is the closest frame that overlaps with targeted segments. The refining of the search space continues and only qualified objects that are inside the search range (i.e., the corresponding HC values are bounded by the targeted segments) are retrieved. Eventually, the search space is finalized (i.e., the inner solid circle), all the inside objects (i.e., o_{32}, o_{40} and o_{51}) are retrieved, and the targeted segment becomes empty. As shown in Figure 9 where the client skips the download of dark frames, the access latency is 7 frames and the tuning time is 6 frames.

Aggressive Approach.
The aggressive approach is different from the conservative approach in refining the size of the search space. As we explained previously, the conservative approach accumulates the knowledge about object distribution gradually, according to the broadcast order of frames. How-

Algorithm 3 Conservative kNN query processing

Input: a query point q, and the requested number of nearest neighbors k;
Output: k nearest neighbors to q;
Procedure:
1: $r :=\infty$; $R := \emptyset$; $F' :=$ NULL; $seg := \{0 - HC_{max}\}$;
2: begin the initial probe and find the pointer P to the first frame F_s;
3: **while** seg is not empty **do**
4: switch to doze mode until frame F_s pointed at by P arrives;
5: **for** $(i := 0; i < n_i; i := i + 1)$ **do**
6: $o'_i :=$ the object represented by HC'_i;
7: **if** $|o'_i, q| < r$ **then**
8: $r :=$ insert(R, o'_i, k);
9: **for** all the data objects obj_i contained in F_s **do**
10: **if** $|obj_i, q| <= r$ **then**
11: retrieve obj_i; $r :=$ insert(R, obj_i, k);
12: refine targeted segment seg based on current search range;
13: **for** $(P := NULL, i := 0; i < n_i; i := i + 1)$ **do**
14: **if** HC'_i is bounded by targeted segment seg **then**
15: $P := P_i$; *break*;
16: return R;

ever, frames close to the query point can provide more precise knowledge about objects distribution that we are interested in than those far away ones. Motivated by this fact, the aggressive approach tries to jump to the frame that is closest to the query point to expedite the finalizing of the search space. Hopefully, the retrieval of some intermediate objects that are falling outside the final search space can be avoided and hence the client energy is saved. However, it may skip some answer objects and the client has to collect them in the next broadcast cycle, which incurs a longer access latency.

As the algorithm is very similar to the conservative approach, we do not list the pseudo-code but use the same running example to illustrate the aggressive kNN processing (also shown in Figure 9). After downloading the index table of the first frame F_1, the client decides to follow P_2 since HC'_2 (i.e., 32) is closest to the query point, compared to other indexed HC values. It skips frames F_2, F_3, F_4 to access F_5 directly. Based on the index table, it detects the existence of objects o_{11}, o_{32}, o_{40}, and o_{51}. Accordingly, a search circle centered at q covering o_{32}, o_{40}, and o_{51} (3NN objects she knows as far) is set as the initial search space. The corresponding targeted segments are set as $\{9-10\}, \{27-28\}, \{31-36\}$, $\{39-40\}$, and $\{43-55\}$. Based on index information associated with frame F_1, segment $\{9-10\}$ containing no existing object can be removed. Since object o_{32} is next to q along the Hilbert curve[8], the remaining search steps are similar as those in the conservative approach, i.e., sequentially retrieving all the data objects located within the search space.

The client downloads object o_{32}, and accesses frames F_6 and F_7. Although the three answer objects are retrieved, the search is not completed since the targeted segments ($\{28-28\}$ and $\{31-31\}$) are not empty. At frame F_7, the client will follow the third pointer P_2 to access frame F_{11}, since neither HC'_0 nor HC'_1 overlaps with targeted segments. The index table shows the next two objects are o_{27} and o_{32}, ruling out the existence of o_{28} and o_{31} and thus terminating the search process. As Figure 9 shows, the access latency is 11 frames and the tuning time is 5 frames.

4.2 Continuous Query Processing on Air

Due to the mobility of users, query points may move continuously, which makes the retrieval of location-based data a challenge. This issue of query continuity is particularly important for navigation and tour guide applications. Repeatedly issuing snapshot window queries or nearest neighbor queries along each moving point is obviously not a feasible strategy. Since the answer to new query points may very likely remain the same as that to a previous query point, there is no need to continuously issue and process the query while moving. One approach to address this issue is to allow a client to issue a *continuous query* by specifying a projected moving trajectory in order to retrieve answers corresponding to an anticipated trajectory in a timely fashion. As a result, answers paired with corresponding segments of the trajectory (where the answers are valid) are returned and cached in the client. Thus, efficient algorithms for processing continuous queries along projected trajectories are very important. Again, our solution aims at reducing energy consumption and access latency. In the following, we present the search algorithms for both continuous window query and continuous nearest neighbor search.

4.2.1 Continuous Window Queries A continuous window (CW) query returns all the objects which are located inside a sliding query window whose center (i.e., query point) lies along a given *query line segment*. Each object is associated with a *validity segment*, i.e., a portion of the query line segment where the object is covered by all the query windows centered along that line. For example, a client plans to drive to Boston along Highway I-93, and wants to locate all the nearby gas stations in advance since she forgot to gas up her car. A continuous window query: "locate all the gas stations within 1 mile from my current location to Boston along Highway I-93" can be issued to retrieve all the qualified gas stations. Unlike snapshot window query, a query point of CW query is not fixed and moves along a projected line segment. Due to the change of query point, the distances between the query point and data objects are no longer fixed. The result of a window query issued at one point may not be the same as that issued at another point, as illustrated in Figure 10. In the figure, a query point moves along a query line segment $\overline{se}$ passing through three distinct points p_1, p_2 and p_3. The query results to window queries issued at p_1, p_2 and p_3 are $\{o_1, o_2, o_3\}$, $\{o_2, o_3\}$, and $\{o_2, o_3, o_4\}$. Here, o_2 and o_3 are the common answer objects to the three adjacent window queries. However, o_1 becomes invalid after p_2 while o_4 becomes valid beyond p_3. This observation implies that each object o has its own validity segment VS_o. In this paper, our continuous window query is defined to include a set of objects, and each object o is associated with a validity segment VS_o. The formal definitions of a continuous window query and a validity segment are given as follows. Here, $W^{w\times l}(p)$ represents a rectangular window centered at point p, having w as width and l as length, and $o \in W^{w\times l}(p)$ means the object o is located inside the window.

Definition 1 Given a continuous window query issued along $\overline{se}$ with $W^{w\times l}$ as the query window, a *validity segment* to an answer object o stands for a subsegment $VS_o \subseteq \overline{se}$ such that $\forall p \in VS_o$, $o \in W^{w\times l}(p)$. □

[8] Since it is impossible to find real nearest object to a given query point q without complete knowledge of object distribution, we treat the object closest to q along the Hilbert curve as the best candidate for its nearest neighbor.

Definition 2 Given a query line segment $\overline{se}$, a query window $W^{w\times l}$, and a set of objects $\mathcal{O}$, a *continuous window query* retrieves all the objects $o \in \mathcal{O}$, such that $\exists p \in \overline{se}$, $o \in W^{w\times l}(p)$. All the qualified objects o, together with their corresponding validity segment VS_o, form the answer set. □

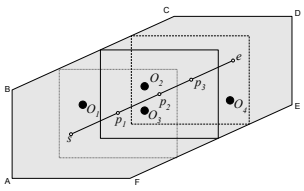


Fig. 10 Illustration of CW Query

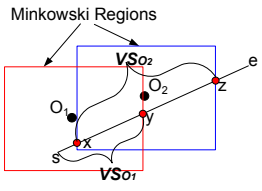


Fig. 11 Minkowski Region and Validity Segment

Minkowski region can be used to determine the validity segment for each answer object. Every object o has a Minkowski region that shares the same size as the query window and centers at o, i.e., $W^{w\times l}(o)$. The intersection of the query line segment with the Minkowski region provides the validity segment of the object o, i.e., $VS_o = \overline{se} \cap W^{w\times l}(o)$. Figure 11 shows the Minkowski regions for objects o_1 and o_2. According to the intersection between Minkowski regions and the query line segment, we can determine $VS_{o_1} = \overline{sy}$ and $VS_{o_2} = \overline{xz}$.

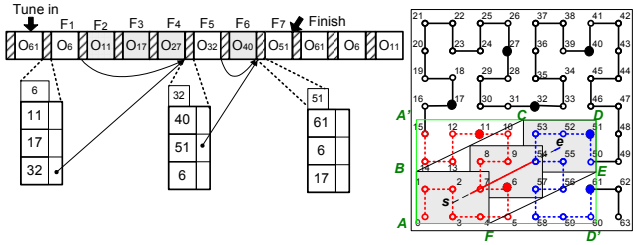


Fig. 12 Example of Continuous Window Query

Due to a large overlap among the results of adjacent window queries, our strategy for processing a CW query is described as follows. Firstly, clients collect all the objects that are covered by query windows issued from points along the line segment. Secondly, for each object, we identify the corresponding validity segment. For the first step, we derive a search range that bounds all the answer objects. Figure 12 shows an illustrative example with a CW query issued along a query line segment $\overline{se}$. By sliding a window along $\overline{se}$, we obtain a polygon ABCDEF that covers all the possible query windows whose centers lie along $\overline{se}$. Instead of finding the targeted segment for a complicated polygon, we construct a minimal bounding window, i.e., rectangle AA'DD', that tightly covers the polygon. As discussed in snapshot window query, the targeted segment set H for the rectangle AA'DD' which contains two segments, $\{0-15\}$ and $\{50-61\}$, can be easily determined. H serves as a guidance for object retrieval, but the real retrieval happens only when the objects are located in the polygon ABCDEF.

Suppose the client tunes into the channel as depicted, the first frame F_1 it receives contains object o_6, inside the polygon ABCDEF. The client retrieves the object and then follows the last index entry to access the index table associated with o_{32}. After accessing F_1, the segment $\{0-15\}$ is shrink to $\{0-5\}$. The client switches to doze mode, and wakes up when frame F_5 is about coming. According to the index table associated with F_5, she skips F_6 and follows the second pointer to access F_7. The search is completed after downloading object o_{51}. This is because 1) the first entry of F_7's index table points to o_{61} which is outside the polygon ABCDEF, indicating a safe removal of the targeted segment $\{50-61\}$; and 2) the second entry of F_7's index table points to o_6 again showing the beginning of a new broadcast cycle and implying that no objects are within the remaining search space $\{0-5\}$. The validity segments of answer objects o_6 and o_{51} can be derived thereafter according to their Minkowski Regions[9]. The access latency is 7 frames and the tuning time is 3 frames. Algorithm 4 lists the pseudo code.

Algorithm 4 Continuous Window Query

Input: a query line segment $\overline{se}$, a specified query window $W^{w\times l}$, sorted HC values of objects;
Output: answer objects together with their corresponding validity segments;
Procedure:
1: determine a polygon $poly := \cup W^{w\times l}(p)$, $\forall p \in \overline{se}$;
2: determine a minimal bounding window W that bounds polygon $poly$;
3: res:=Window Query(W); /* only those objects falling inside $poly$ are retrieved */
4: $res' := \emptyset$;
5: **for** each object $o \in res$ **do**
6: find its Minkowski region M_o, i.e., $M_o := W^{w\times l}(o)$;
7: find its validity segment VS_o, i.e., $VS_o := M_o \cap \overline{se}$;
8: res':=$res' \cup \{o, VS_o\}$;
9: return res';

4.2.2 Continuous Nearest Neighbor Queries Continuous-nearest-neighbor (CNN) query finds a set of nearest neighbors corresponding to every point in a given *query line segment*. Example queries like "locate all the nearest Chinese restaurants along the Fifth Avenue from my current position to the Central Park" issued by a visitor touring in Manhattan. Similar to kNN search, one challenge faced in processing CNN query is how to determine a search space large enough to contain all the results. In addition, unlike kNN search, CNN extends

[9] The green and red segments along $\overline{se}$ (shown in Figure 12) represent the validity segments of objects o_6 and o_{51} respectively, i.e., VS_{o_6} and $VS_{o_{51}}$.

the query point to a query line segment, which makes the search algorithms different and challenging.

It has been pointed out that the answer set to a CNN query contains a set of objects, denoted as o_1, o_2, $\cdots$, o_n, which partition the corresponding query line segment into n smaller segments l_i ($i \in [1, n]$) [32]. Each answer object o_i *dominates* a corresponding segment l_i, i.e., o_i is the nearest neighbor of any query point lying on segment l_i. This is similar to the validity segments returned by a continuous window query. An example is depicted in Figure 13. Objects o_1, o_2, and o_4 form the CNN answer set to the query line $\overline{se}$. o_1 dominates the shadowed line segment $\overline{sp_1}$, which means o_1 is the nearest neighbor to any point lying along the segment $\overline{sp_1}$. Similarly, o_2 dominates $\overline{p_1p_2}$ and o_4 dominates $\overline{p_2e}$. p_1 and p_2 are called *split points* [34] since they are the points at which the nearest objects along the line segment change. Other than finding CNN result objects, determining split points is clearly not trivial. In addition, as the arrival of objects depends on the broadcast schedule, the search space refinement and object filtering render CNN query processing particularly challenging.

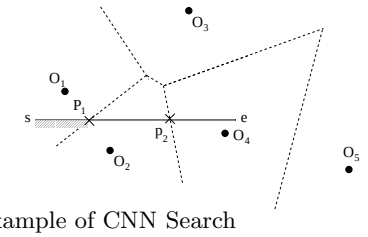


Fig. 13 Example of CNN Search

Our CNN query processing is briefly described as follows. First, the initial search space is set to a space covering all the available data objects.The size is thereafter reduced as more knowledge about the data distribution is accumulated. The adjustment of the search space is based on Heuristic 1 and is detailed in Algorithm 5. Only those frames with objects fallen in the (immediate) search range are retrieved. However, not all the retrieved data objects can contribute to the final answer set. In order to filter out the unqualified objects quickly, Heuristic 2 is used to simplify the objects checking process and hence reducing the computational cost. The search continues until the search range is finalized and all the objects inside it are downloaded and checked. The search algorithm for CNN queries is summarized in Algorithm 6. Table 3 defines all the notations used in the following description.

Heuristic HR1. Given a query segment $\overline{se}$ and a corresponding set of split points $SL(\overline{se})$, the maximal distance between any point p ($p \in \overline{se}$) and $NN(p)$ is bounded by D_{max}, where $D_{max} = MAX(|sp, sp.NN|)$, $\forall sp \in SL(\overline{se})$. Take Figure 14 as an example, and assume $SL(\overline{se})$ currently contains two split points, sp_1 and sp_2. For $\forall p \in \overline{ssp_1}$, $|p, NN(p)|$ should not be greater than $MAX(r_1, r_2)$, since $|p, o_1| \leq MAX(r_1, r_2)$ and $|p, NN(p)| \leq |p, o_1|$. Similarly, it is guaranteed that $|p', NN(p')| \leq MAX(r_3, r_4)$ and $|p'', NN(p'')| \leq MAX(r_5, r_6)$, with $p' \in \overline{sp_1sp_2}$ and $p'' \in \overline{sp_2e}$. As a result, $|q, NN(q)|$ ($q \in \overline{se}$) is bounded by r_3, the largest distance among all split points and their corresponding nearest neighbors.

Notation	Description
$\|p_1, p_2\|$	Euclidean distance between points p_1 and p_2
$CNN(\overline{se})$	the answer set containing all the nearest neighbors to any point in the segment $\overline{se}$
$SL(\overline{se})$	the set of split points found so far for the segment $\overline{se}$
$p.NN$	the nearest-neighbor found so far to point p
$NN(p)$	the real nearest-neighbor to the point p

Table 3 Terminology Definition

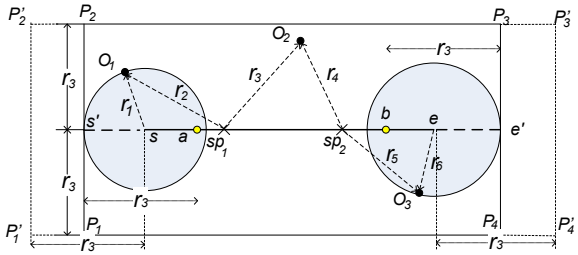


Fig. 14 Approximation of CNN Search Range

Algorithm 5 RefineCNNSearchRange

Input: query line segment $\overline{se}$, $SL(\overline{se})$;
Output: a search range;
Procedure:
1: $D_{max} := 0$;
2: **for** each point sp in $SL(\overline{se})$ **do**
3: $dis := |sp, sp.NN|$;
4: **if** $dis > D_{max}$ **then**
5: $D_{max} := dis$;
6: extend segment $\overline{se}$ to $\overline{se'}$, with $|ee'| := |e, e.NN|$;
7: extend segment $\overline{es}$ to $\overline{es'}$, with $|ss'| := |s, s.NN|$;
8: draw a line l passing s' and perpendicular to line $\overline{se}$;
9: find two points P_1 and P_2 on l such that $|P_1, s'| := D_{max}$ and $|P_2, s'| := D_{max}$;
10: draw a line l' passing e' and perpendicular to line $\overline{se}$;
11: find two points P_3 and P_4 on l' such that $|P_3, e'| := D_{max}$ and $|P_4, e'| := D_{max}$;
12: return the rectangle bounded by P_1, P_2, P_3 and P_4;

Based on HR1, a circle $cir(a, D_{max})$, which is centered at a point a ($\in \overline{se}$) and has D_{max} as radius, is guaranteed to bound the nearest neighbor to point a. As a result, a straightforward approach to approximate the search range is to bound all the circles $cir(a, D_{max})$ for all the points a along the query line segment, which forms a pill-shaped region. Further, we can use the rectangle $P'_1P'_2P'_3P'_4$ as shown in Figure 14 to bound the search area. In addition, the rectangle can be trimmed

(as shown in Algorithm 5). Its correctness is warranted by Lemma 1.

Lemma 1 The search range approximated by Algorithm 5 is guaranteed to enclose all the nearest neighbors to all the points along a given query line segment. □

Proof: Without loss of generality, we use the one shown in Figure 14 as a running example. In order to prove that rectangle $P_1P_2P_3P_4$ is big enough to cover all the nearest neighbors, we first need to prove that the nearest neighbor to any point along $\overline{se}$ will not fall inside rectangle bounded by $P'_1P'_2P_1P_2$.

We assume points a, b are two points on $\overline{se}$ such that $|\overline{s'a}|=|\overline{be'}|=D_{max}$. In other words, the points p lying along $\overline{sa}$ (or $\overline{be}$) are the only points whose corresponding circles $cir(p, D_{max})$ are not bounded by the search range $P_1P_2P_3P_4$. Given a point $p \in \overline{sa}$, we assume its nearest neighbor $NN(p)$ is outside rectangle $P_1P_2P_3P_4$. As a result, the distance between p and o_1 (i.e., $s.NN$) must be larger than that between p and $NN(p)$. On the other hand, points o_1, s, and p form a triangle, and $|o_1,p|^2=|s,o_1|^2+|s,p|^2$-2 $\cdot |s,o_1| \cdot |s,p| \cdot cos(\angle o_1sp)$.

If $cos(\angle o_1sp) \neq -1$, $|o_1,p| \leq (|s,o_1|+|s,p|)^2$. It is obvious that $(|s,o_1|+|s,p|) = |p,s'| \geq |o_1,p|$. As $|p,NN(p)| \geq |p,s'|$, $|p,NN(p)| \geq |o_1,p|$. Consequently, the assumption that $NN(p)$ is outside rectangle $P_1P_2P_3P_4$ is not valid.

Otherwise, $cos(\angle o_1sp) = -1$, which means o_1 overlpas with s'. Obviously, $|p,s'|$ is the lower bound of the distance between p and any point falling inside rectangle $P'_1P'_2P_1P_2$ (i.e.,$|p,s'| \leq |p,NN(p)|$). Consequently, the assumption that $NN(p)$ is outside rectangle $P_1P_2P_3P_4$ is not valid.

Similarly, we can prove that the nearest neighbor to any point along line $\overline{be}$ must be inside rectangle $P_1P_2P_3P_4$. As a result, the correctness of algorithm 5 is proved. ■

Heuristic HR2. Given a query segment $\overline{se}$ and a set of split points $SL(\overline{se})$, a new object o' belongs to $CNN(\overline{se})$ if and only if $\exists sp \in SL(\overline{se}), |sp,sp.NN| > |sp,o'|$. □

The above heuristic has been proved in [34]. All the objects that do not have a closer distance to any split point than the distance between the split point and its current nearest neighbor are not qualified and can be safely discarded. HR2 serves as the objects retrieval guidance and tries to save the client's energy by skipping unqualified objects.

Based on heuristics HR1 and HR2, Algorithm 6 shows the pseudo code of the query processing algorithm for CNN queries. Here we use a simple example (as shown in Figure 15) to illustrate the CNN search process. Suppose a client who tunes into the channel as depicted wants to find all the nearest neighbors (e.g., gas stations) along its motion path (represented by a horizontal line segment $\overline{se}$). According to the index table of F_1, the client is aware of the existence of objects o_6, o_{11}, o_{17}, and o_{32} and thus can determine o_{32} is the current nearest neighbor which dominates the entire query line segment. Accordingly, the client can approximate the search range based on current D_{max}, i.e., $|o_{32},e|$. The light gray rectangle depicted in the figure is the corresponding search range.

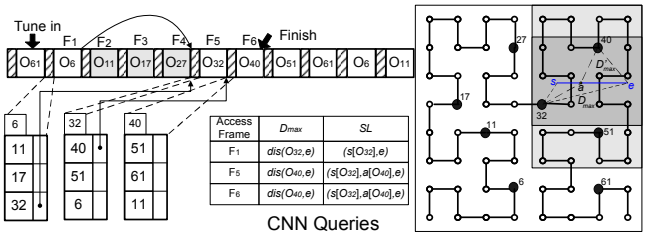


Fig. 15 CNN Query Processing

Thereafter, the client skips frames F_2, F_3, and F_4 (since they are all outside the current search range), and then accesses frame F_5. According to the index table associated with frame F_5, object o_{40} is identified to be closer to point e than e's current nearest neighbor (i.e., o_{32}). As a result, o_{40} becomes a new potential answer object and a new split point (i.e., point a) is introduced. Now, the current answer set contains two objects o_{32} and o_{40}, that are NN objects to segment $\overline{sa}$ and segment $\overline{ae}$ respectively. The search range is shrunk to the dark gray rectangle. The search process is completed when the frame F_6 is downloaded since all the objects inside the search range are retrieved. Finally, the access latency is 6 frames, and the tuning time is 3 frames.

Algorithm 6 CNN Query Processing

Input: a query segment $\overline{se}$;
Output: $CNN(se)$;
Procedure:
1: $res := \emptyset$; $SL := \emptyset$; begin the initial probe and retrieve frame F_s;
2: **while** $F_s \neq$ NULL **do**
3: **for** all the pointers P_i in the index table of F_s **do**
4: o'_i := the object represented by HC'_i;
5: **if** $\exists p \in SL$, $|o'_i,p| < |p,p.NN|$ **then**
6: addNewNN(SL,o'_i);
7: **for** all the data objects obj_i within F_s **do**
8: **if** $\exists p \in SL$, $|obj_i,p| \leq |p,p.NN|$ **then**
9: addNewNN(SL,obj_i); download object obj_i;
10: r := RefineCNNSearchRange($\overline{se}$, SL);
11: **for** all the pointers P_i in the index table of F_s **do**
12: **if** $(HC'_i < HC'_{(i+1) \bmod n_i})$ and ($\{HC'_i$-$HC'_{(i+1) \bmod n_i}\}$ overlaps with r) **then**
13: F' := the frame pointed at by the pointer P_i; break;
14: **else if** $(HC'_i > HC'_{(i+1) \bmod n_i})$ and ($\{HC'_i$-$HC_{max}\}$ or $\{0$-$HC'_{(i+1) \bmod n_i}\}$ overlaps with r) **then**
15: F' := the frame pointed at by the pointer P_i; break;
16: $F_s := F'$; $F' :=$ NULL;
17: **for** all split points sp in SL **do**
18: $res := res \cup \{sp.NN\}$;
19: return res;

4.3 Discussion: Queries on Multiple Data Types

Although our algorithms presented in this section are for homogeneous data types, all of them can be naturally extended to support heterogeneous data types if the query does not depend on the proximity of the data objects from different types. For example "return all the hotels, shopping malls and Italian restaurants within 1000 meters to the VLDB 2008 conference venue". Here we would like to highlight a point about NN type queries (such as kNN search and CNN search) under the integrated approach. Originally, the index entry $\langle HC'_j, P_j \rangle$ maintains the smallest HC value of data objects contained in the frame that P_j points to, and it is for sure that an object with the HC value HC'_j is present. The search can utilize this information to refine the search range. However, data objects of different types might be broadcast in one frame in the integrated approach. Consequently, a kNN search of a particular data type i cannot use the HC'_j value directly as it is not guaranteed that the data with HC value HC'_j actually belongs to the data type T_i. One possible solution to this issue is to ignore the search range refinement step, as shown in lines 5-8 in Algorithm 3. Alternatively, instead of keeping $\langle HC'_j, P_j \rangle$ pairs in the index table, we can store $\langle HC^1_j, HC^2_j, \cdots, P_j \rangle$ with HC^i_j representing the HC value of the first object corresponding to the data type T_j contained in the target frame. In addition, more complex queries such as locating the top-three closest apartments with kindergartens within 500 meters involve the spatial join. We leave the detailed investigation of these queries in our future work.

5 Performance Evaluation

This section evaluates the performance of DSI for supporting various spatial queries. The queries to be evaluated include 1) point queries using EEF, 2) snapshot window queries and snapshot k nearest neighbor (kNN) queries, and 3) continuous window queries and continuous NN queries. In order to show that tree-based index structures are inefficient for broadcast environments, we develop an R-tree-based air index to compare with DSI[10]. To construct an R-tree-based air index, we employ the STR packing scheme [22] that has been shown to provide optimal search performance. Further, to reduce the initial probing cost, we replicate the top portion of the R-tree evenly in the broadcast channel based on the well-known distributed data organization scheme [18]. The replication level is optimized according to the analytical model presented in [18]. Without loss of generality, we assume each index node occupies one packet, which in turn determines the fanout of the R-tree. We label this R-tree-based air index as **STR R-Tree**.

[10] R-tree is considered the best index structure for low-dimensional spatial data.

Our design of DSI is highly flexible and configurable by adjusting two major parameters, namely, exponential base r and object factor n_o. In this evaluation, we fix r at 2 while varying n_o. We leave the study on combinations of r and n_o for future research. The impact of n_o is mainly on the number of frames, n_F $(= \lceil N/n_o \rceil)$, in the broadcast, which in turn affects the number of index table entries (i.e., $n_i = \lfloor log_2(n_F) \rfloor = \lfloor log_2(\lceil N/n_o \rceil) \rfloor$) carried by a frame. In this paper, we simply allocate one packet for the index table associated with each frame, which is consistent with that for R-tree. Thereafter, the total number of frames n_F and object factor n_o can be derived.

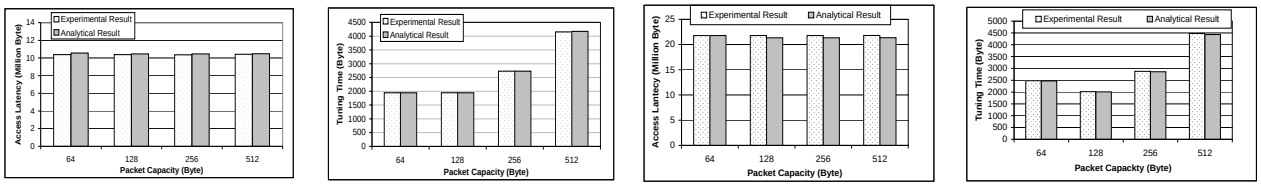
The performance evaluation is based on simulations. Our simulator is built with CSIM18 [29]. It consists of a server, a client and a broadcast channel between them[11]. The server disseminates an entire data set on the broadcast channel repeatedly. A client processes one query at a time. At the time the client issues a query, it tunes into the broadcast channel to collect the requested spatial objects. After the query is processed, the client becomes idle and is reactivated when the next query arrives.

For each set of experiments, the simulation runs for $100,000$ randomly issued queries based on both the UNIFORM and the REAL datasets. The results presented in this section are the average of $100,000$ queries. UNIFORM is a synthetic dataset containing $20,000$ points uniformly distributed in a unit square. REAL is a real dataset containing over $40,000$ schools (point locations) in US obtained from Tiger/Line [9]. A coordinate is represented by a pair of 4-byte float-point numbers, and an HC value for a 2-D spatial object is represented by an 8-byte integer. The size of a spatial object is fixed at $1,024$ bytes.

The query performance metrics include access latency and tuning time. Suppose the bandwidth of the broadcast channel is fixed, we simply report the number of bytes transferred over the broadcast channel in place of real clock time. Firstly, this does not involve any complicated unit conversion from bytes into time units. Secondly, we avoid using the number of packets for comparison since the packet capacities in our experiments are varied to model different real life network payload characteristics.

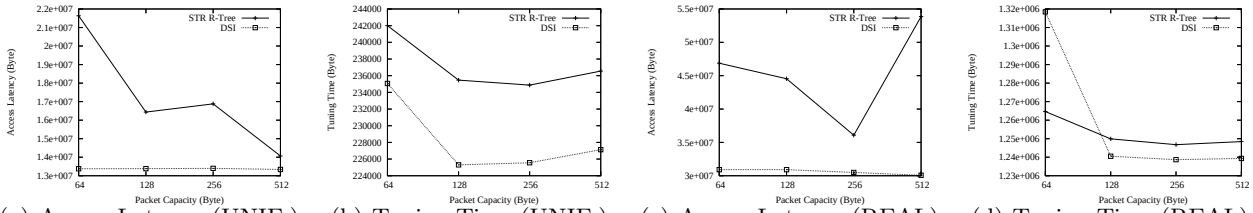
Our first set of experiments studies the performance of DSI and R-tree for different spatial queries, namely point queries, snapshot window queries, snapshot kNN query, continuous window queries and continuous nearest neighbor searches. Our second set of experiments evaluates the computational cost and power consumption of different indexes. All these experiments are based on a reliable wireless environment, i.e., the percentage of link errors $\theta = 0$. Thereafter, we conduct another set of

[11] In a broadcast system, the number of clients will not affect the search process conducted by each individual client. For simplicity, we only simulate one client in this model.



(a) Access Latency (UNIF.) (b) Tuning Time (UNIF.) (c) Access Latency (REAL) (d) Tuning Time (REAL)

Fig. 16 Performance of Point Query vs. Packet Capacity



(a) Access Latency (UNIF.) (b) Tuning Time (UNIF.) (c) Access Latency (REAL) (d) Tuning Time (REAL)

Fig. 17 Performance of Window Queries vs. Packet Capacity ($WSR = 0.1$)

experiments to simulate an error-prone environment in which packet loss happens in order to demonstrate the superior resilience of DSI to errors. All the above simulations are conducted based on the assumption that a homogeneous data type is considered. In order to evaluate the performance of DSI under multi-data type environments, we conduct the last set of experiments to evaluate the performance of window queries and continuous window queries under the sequential approach and the integrated approach.

5.1 Point Queries

The first set of experiment evaluates DSI for point queries in terms of access latency and tuning time. We vary the packet size from 64 up to 512 bytes. We also use the simulation results to validate our analytical model. The experimental results of the UNIFORM and REAL datasets are depicted in Figure 16. The variation of packet sizes does not affect the access latency since the increase of packet sizes does not extend the broadcast cycle much. From the plots, we observe the consistency between the simulation and analytical results, which demonstrates the correctness of our analytical model.

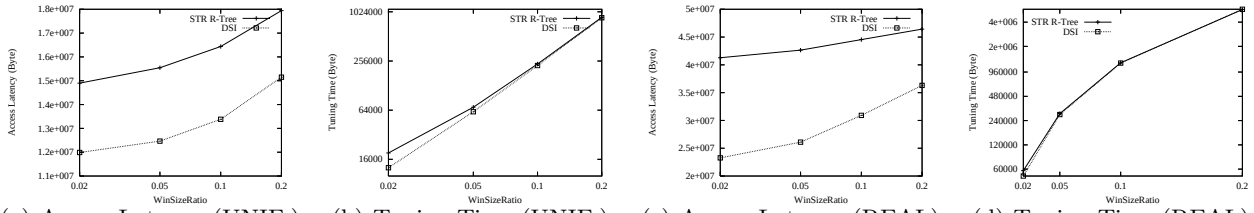
5.2 Snapshot Queries

Next, we conduct experiments to evaluate the performance of DSI and R-tree in answering snapshot window and kNN queries. For window query, we introduce a parameter *WindowSizeRatio (WSR)*, which is defined as the ratio of the side length of the query window to that of the whole search space (default = 0.1), to control the sizes of the query windows. For kNN searches, we adjust parameter k to control the size of the answer set.

5.2.1 Snapshot Window Queries In this experiment set, we evaluate the performance of DSI and R-tree for supporting window queries. In the first experiment, we fix *WSR* at 0.1 and vary packet capacity from 64 to 512 bytes. The results are plotted in Figure 17. It can be observed that DSI is superior to R-tree in terms of access latency. As the packet capacity increases, the access latency of DSI remains small and stable while that of R-tree drops. On average, DSI requires only 79% and 66% of the access latency of R-tree for the UNIFORM and REAL datasets, respectively. This is because DSI can start the search right after the client tunes into the channel, and the access latency is the duration of time from the starting frame to the last retrieved frame. Although the capacity changes, the time duration changes in terms of the number of packets but not the number of bytes. As a result, the access latency of DSI is very low and stable. On the other hand, window search performance of R-tree is affected by the relationship between MBRs and the query windows. As packet capacity changes, the fanout and hence MBRs change. The search path from the root node down to the leaf nodes changes accordingly and thus the performance varies.

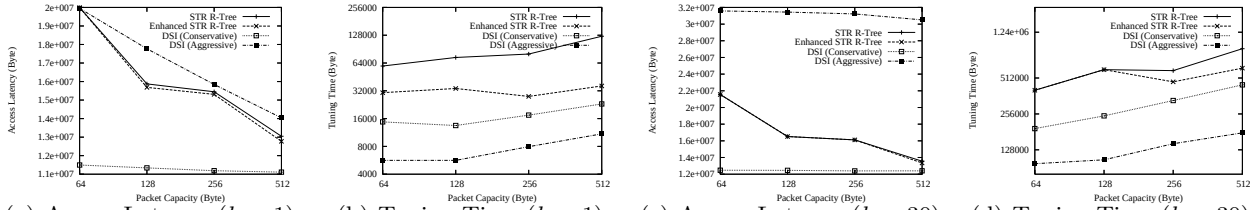
Besides, DSI performs better than R-tree in terms of tuning time. This is attributed to the compactness of DSI, which indexes 1-D HC values instead of 2-D MBRs as R-tree does. In addition, MBRs may overlap with query windows which might not contain any answer object. On average, DSI consumes only 96% of the tuning time of R-tree for the UNIFORM dataset and 90% of the tuning time of R-tree for the REAL dataset.

To study the impact of query window size, we fix packet capacity, denoted as C, to 128 bytes and vary *WSR* from 0.02, to 0.05, to 0.1, and to 0.2 in the second set of experiments. The results for the UNIFORM and REAL datasets are depicted in Figure 18. Again, DSI is observed to be outperforming R-tree in terms of both the access time and the tuning time. In comparison with R-tree, DSI only consumes 95% of the tuning time and requires 66% of the access latency on average for the REAL dataset. As the window size becomes larger, more objects have to be retrieved and hence the retrieval cost



(a) Access Latency (UNIF.) (b) Tuning Time (UNIF.) (c) Access Latency (REAL) (d) Tuning Time (REAL)

Fig. 18 Performance of Window Queries vs. WSR (C=128Bytes)



(a) Access Latency ($k = 1$) (b) Tuning Time ($k = 1$) (c) Access Latency ($k = 30$) (d) Tuning Time ($k = 30$)

Fig. 19 Performance of kNN Queries vs. Packet Capacity (UNIFORM)

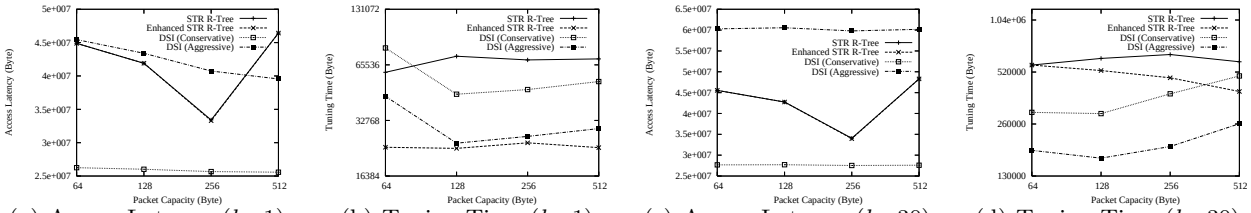
of the qualified objects (which is independent of the index structure and search algorithm) dominates the tuning time. Consequently, both DSI and R-tree have similar tuning time performance. The advantage of DSI under the UNIFORM dataset is even more significant, as shown in Figure 18(a) and Figure 18(b). For the same sets of window queries, by using DSI, clients can reduce 23% of the tuning time and 29% of the access time from that of R-tree.

5.2.2 Snapshot K Nearest Neighbor Queries In this section, the performance of DSI and R-tree algorithms for supporting kNN queries is evaluated. For R-tree index, the kNN search algorithm would visit index nodes and objects sequentially as backtracking is not feasible on the broadcast. This certainly results in a considerably long tuning time especially when the result objects are located in the later part of the broadcast. However, if the client knows that there are at least k objects in the later part of the broadcast that are closer to the query point than the currently found ones, they can safely skip the downloading of the intermediate objects currently located. This observation motivates the design of the enhanced kNN search algorithm (please refer to [12] for details). It requires each index node to carry a count of the underlying objects ("object count"). Thus, clients do not blindly download intermediate objects. In our experiment, we include this enhanced kNN search algorithm for comparison. Instead of including "object count" in every index node, we continue to use STR-tree and use the level of the index node and the fanout of the index (which is fixed) to determine object counts at runtime. We label this extended algorithm as **Enhanced STR R-tree**. The performance shown in this section is actually slightly better than its real performance, because the overhead of "object count" is not included in our measurement.

Firstly, we examine the performance of the DSI and R-tree algorithms for 1NN (i.e., k=1) and 30NN (i.e., k=30) queries with different packet capacities. Figure 19 plots the performance for the UNIFORM dataset. At first sight, the aggressive approach is the most energy efficient. It dramatically saves more tuning time than both the conservative approach and R-tree-based approaches. This aligns with our expectation since the aggressive approach skips the objects located far away from the query point. Therefore, the search range could be shrunk quickly and hence fewer false result objects are downloaded. For example, we track the number of objects retrievals incurred for the UNIFORM dataset (C = 64 bytes, and k = 30) and find that on average only 75 objects are retrieved by the aggressive approach, compared with 223 by the conservative approach. Meanwhile, the R-tree algorithms scan a number of index nodes and some additional objects to determine the final result, making them less competitive than DSI. However, the aggressive approach suffers from a much longer access latency. This is because the kNN search process is postponed, i.e., the client might miss some answer objects in the current broadcast cycle, thereby forcing it to download them in the next cycle.

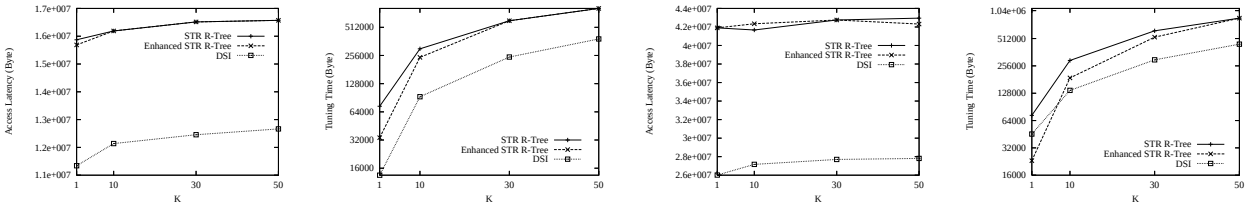
In summary, when k = 1, conservative approach incurs only 71% of R-tree's access latency[12], and it consumes 11% of R-tree's and 53% of Enhanced R-tree's tuning time, respectively. When k is set to 30, on average conservative approach incurs 75% of R-tree's access latency for UNIFORM dataset. Meanwhile, it consumes 49% of R-tree's and 58% of Enhanced R-tree's tuning time, respectively. Since the conservative approach achieves a good balance between the access latency performance and the tuning time performance, we use it as the default algorithm for kNN search in the following discussion.

[12] The Enhanced kNN search algorithm only improves tuning time performance, but not access latency.



(a) Access Latency (k=1) (b) Tuning Time (k=1) (c) Access Latency (k=30) (d) Tuning Time (k=30)

Fig. 20 Performance of kNN Queries vs. Packet Capacity (REAL)



(a) Access Latency (UNIF.) (b) Tuning Time (UNIF.) (c) Access Latency (REAL) (d) Tuning Time (REAL)

Fig. 21 Performance of k-NN Queries vs. k (C=128Bytes)

For the REAL dataset, the performance difference between R-tree and DSI (conservative approach) narrows, as shown in Figure 20. This is because R-tree ensures high spatial locality. When the distribution of objects is skewed, the search space represented by high-level index nodes which contain farther objects from a query point can be pruned efficiently, thereby improving the search performance. For NN search, Enhanced R-tree performs better than DSI in terms of tuning time. However, DSI still provides the best access latency. For NN search, it requires 63% of the access latency of R-tree. When k increases to 30, DSI has a better performance than both R-tree and Enhanced R-tree. DSI consumes 69% of R-tree's and 92% of Enhanced R-tree's tuning time, respectively. This is because R-tree would have different search paths for scattered result objects. Meanwhile, for DSI, the proximity of closely located objects is preserved by space-filling curve, which allows search to be constrained within a small portion of the broadcast. In addition to the tuning time, DSI also saves 34% of access latency compared to R-tree.

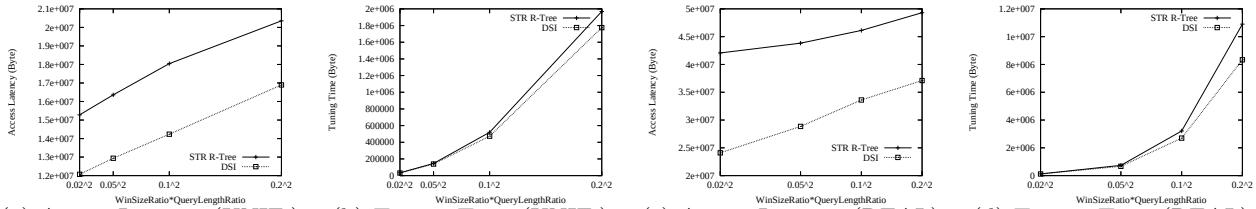
Compared with window queries, the advantages of DSI over R-tree are even more obvious for supporting kNN queries. Our conservative search approach refines the search space gradually while achieving energy efficiency by moving close to the query point via EEF. Next, we vary k from 1 to 50 to examine the impact of the number of nearest neighbors on the performance of spatial air indexes. Figure 21 shows the results for both the UNIFORM and REAL datasets. As expected, DSI performs the best for almost all the cases in both access latency and tuning time.

5.3 Continuous Queries

In this section, we evaluate DSI and R-tree in answering continuous queries, including continuous window queries and continuous nearest neighbor searches, issued from clients when they are moving. A parameter *QueryLength-Ration (QLR)* is introduced to represent the length of the projected linear trajectory (i.e., the query line segment). It is defined as the ratio of the length of the query segment to the side length of the whole search space (default = 0.1). The starting point of a moving trajectory is randomly selected, and the corresponding direction of the movement is uniformly distributed between 0 and $\pi/2$. A client is assumed to issue either a continuous window query, whose size is controlled by WSR, or a continuous nearest neighbor query at the starting point.

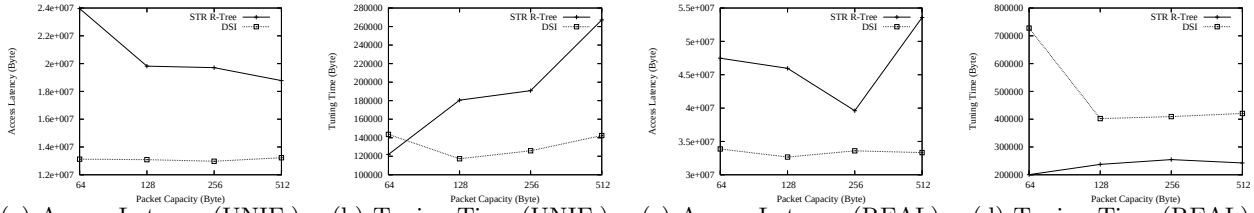
5.3.1 Continuous Window Queries First, the performance of different indexes supporting continuous window queries is evaluated. Different *WSR* values are used: 0.02, 0.05, 0.1, and 0.2. The length of the client moving trajectory is varied via *QLR*. To simplify our experiments, we assume that *WSR* and *QLR* share the same value instead of evaluting all the possible combinations of the two parameter values. In addition, we fix the packet capacity C at 128 bytes.

Figure 22 depicts the simulation result under different configurations of *WSR* and *QLR* values for both the UNIFORM and REAL datasets. Due to the compactness of the index tables and its distributed nature, DSI incurs a much shorter access latency than R-tree. Consider the UNIFORM dataset. DSI takes only 80% of R-tree's access latency. The improvement is even more significant for REAL dataset. DSI incurs only 67% of R-tree's access latency. In terms of tuning time performance, DSI for continuous window queries still performs better than R-tree, even though with a smaller margin. It consumes 94% and 87% R-tree's tuning time for the REAL and UNIFORM datasets, respectively. Compared to snapshot window queries, DSI is less competitive in tuning time. This is because DSI approximates search range as



(a) Access Latency (UNIF.) (b) Tuning Time (UNIF.) (c) Access Latency (REAL) (d) Tuning Time (REAL)

Fig. 22 Performance of CW Search vs. ($WSR \times QLR$) (C=128 Bytes)

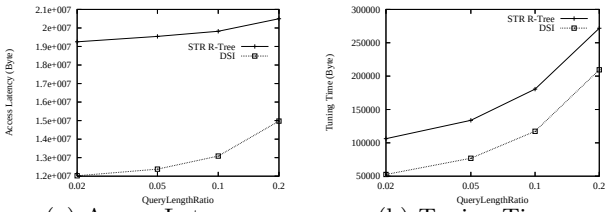


(a) Access Latency (UNIF.) (b) Tuning Time (UNIF.) (c) Access Latency (REAL) (d) Tuning Time (REAL)

Fig. 23 Performance of CNN Search vs. Packet Capacity (REAL: $QLR = 0.1$)

a rectangle based on the given query line segment, which actually takes a large area into consideration and hence introduces some extra overhead.

5.3.2 Continuous NN Queries To evaluate the performance of DSI and R-tree for supporting continuous NN queries, we first fix the length of the query line segment, i.e., QLR, and vary the packet capacity C from 64 bytes to 512 bytes. The result is plotted in Figure 23. DSI is shown to incur a shorter access latency for both the UNIFORM and REAL datasets than R-tree, since DSI enables clients to start the query processing almost right after they tune into the channel while R-tree needs clients to wait for the arrival of the root index node to start the search. On average, DSI requires 64% and 72% of R-tree's access latency under the UNIFORM and REAL datasets, respectively. In terms of tuning time, DSI demonstrates its strength for the UNIFORM dataset by only consuming 75% of R-tree's tuning time. However, R-tree outperforms DSI for the REAL dataset as R-tree benefits more from the skew distribution.



(a) Access Latency (b) Tuning Time

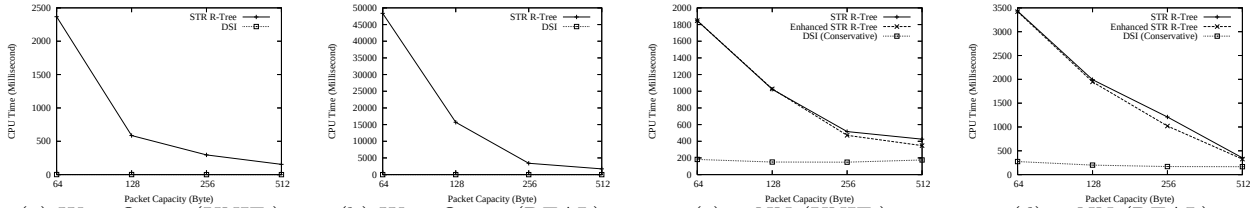
Fig. 24 Performance of CNN Search vs. QLR (UNIFORM: C=128 Bytes)

Next, we evaluate the performance of DSI and R-tree under different settings of QLR. Figure 24 shows the performance for the UNIFORM dataset by fixing the packet capacity C at 128 bytes and varying QLR from 0.02 to 0.2. As expected, as QLR increases, more result objects will be collected. Therefore, more energy is consumed. DSI performs consistently better than R-tree in terms of tuning time and access latency. On average, DSI only requires 62% of tuning time and 66% of R-tree's access latency. For the REAL dataset, DSI does not perform as well as R-tree in terms of tuning time due to the skewed object distribution. However, it incurs considerably less access time, i.e., 72% of R-tree's access latency.

R-tree demonstrates a better performance in tuning time when the distribution of objects is skewed for CNN queries. As we explained above, R-tree clusters objects according to spatial locality and thus allows quick search space pruning if certain tree branches of the objects do not contribute to the query result. On the other hand, the search algorithm based on DSI approximates the search range based on the query line segment and all the candidates found so far. Therefore, it could be very likely that some false results are downloaded before the actual ones are found, thereby introducing a longer tuning time than R-tree. On the other hand, due to compactness and distribution of the index tables, the access time of DSI remains the lowest.

5.4 Computational Cost

Besides the tuning time performance which reflects the number of data objects that a client has to retrieve in order to finish a query, query processing performed at mobile clients also consumes power resource. In this section, we evaluate the computational cost of DSI and R-trees in terms of the CPU time for answering queries. The evaluation is based on a Generic SUN4U SPARC with 4G main memory. Although the SUN machine is much more powerful than existing handheld devices, the result provides good insights and serves as a good reference. In addition, with the trend of rapid technology advances, it is not surprising to have powerful mobile devices with similar computational capability in the future. We conduct experiments on snapshot window queries and snapshot



(a) Win. Query (UNIF.) (b) Win. Query (REAL) (c) 30NN (UNIF.) (d) 30NN (REAL)

Fig. 25 CPU Time of Window Query and 30NN Searches vs. Packet Capacity ($WSR = 0.1$)

kNN queries with various packet capacities. The results are shown in Figure 25.

From the figures, DSI is shown to be superior to R-tree in terms of CPU time for most cases. For window queries, the advantage of DSI is distinct. For all the cases, DSI finishes window queries within 1 millisecond under the UNIFORM dataset that is only 1% of the CPU time of R-tree. For the REAL dataset, DSI requires again only 1% of R-tree's CPU time. For 30NN queries, the superiority of DSI is still very obvious. DSI requires only 24% and 27% of enhanced R-tree and R-tree's CPU time for the UNIFORM dataset, respectively, and it also significantly outperforms Enhanced R-tree and R-tree for the REAL dataset.

5.5 Power Consumption

Conventionally, tuning time is used in the literature to evaluate power consumption. Since it is well-known that mobile devices consume much more energy in the active mode than in the doze mode, an energy-conserving index usually can reduce the tuning time. However, a shorter tuning time does not necessarily save energy due to frequent *switching operations* (i.e., switch between active mode and doze mode), which also incur significant energy cost [35]. In order to fully demonstrate the advantage of DSI, we conduct a simulation to evaluate the power consumption in different approaches.

Different components in mobile devices have different power consumption [1,19]. For example, a processor (StrongARM SA-1100) consumes 0.05mW at doze mode but 200mW at normal mode, a RangeLAN2 PC card consumes 25mW at doze mode, 750mW at receive mode, and 1500mW at transmit mode, and a μ-blox GPS-MS1E consumes 33mW at doze mode and 462mW at normal mode. Based on these data, we can approximate the ratio of power consumed in active mode to that consumed in doze mode, denoted as e, based on $\frac{200+750+462}{0.05+25+33} = 24.34$. Consequently, we can take the power consumed at doze mode as the unit power consumption (denoted as ρ), and power consumed at active mode is $(e \cdot \rho)$, i.e., 24.34 units.

Let P_{on} and P_{off} denote the power consumption in active and doze modes, respectively. Also, let T_{on}, T_{off} and T_{set} denote the time spent in active, doze and setup (i.e., switching) during query processing[13]. Finally, we use N_{swi} to denote the number of switches incurred. We further assume P_{swi}, the power consumption of a switch operation, equals P_{on}. The total power consumption can be derived based on Equation 6.

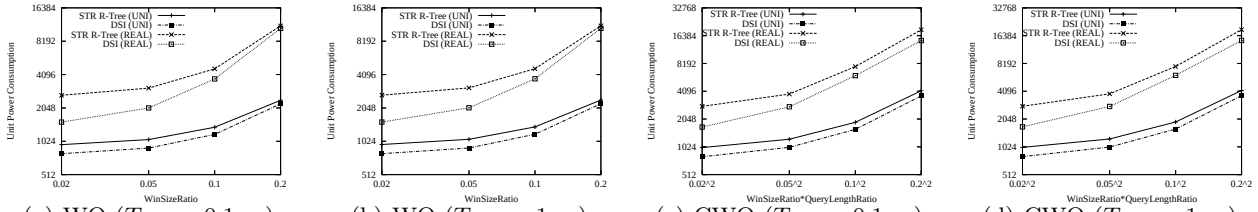
$$\begin{aligned}\mathcal{P} &= T_{on} \cdot P_{on} + T_{set} \cdot N_{swi} \cdot P_{swi} + T_{off} \cdot P_{off} \\ &= T_{on} \cdot (e \cdot \rho) + T_{set} \cdot N_{swi} \cdot (e \cdot \rho) + T_{off} \cdot \rho \quad (6)\end{aligned}$$

It is observed that the typical setup time for a mobile device to start its radio or to tune into active mode is in the order of $100\mu s$ [35]. Since the setup time is device dependent, we vary the setup time, ranging from $100\mu s$ to $1ms$, in our experiments. In order to approximate the time needed to retrieve data objects/index information, we further assume the bandwidth B of the broadcast channel is 128kbps. Figure 26 presents the power consumed of window queries and continuous window searches for both the UNIFORM and REAL datasets. As depicted, DSI saves more power and outperforms R-tree significantly. When $T_{set} = 0.1ms$, DSI consumes only 86.4% and 83.3% of R-tree's power consumption for window query and continuous window query, respectively, under the UNIFORM dataset. For the REAL dataset, DSI consumes only 74.9% and 72.2% of R-tree's power consumption for window query and continuous window query. This is due to more switches occurred in R-tree search algorithms than in DSI algorithms. We further observe that the advantage of DSI in terms of energy-saving is more significant when WSR is smaller (e.g., 0.02). This is because as WSR becomes larger, the answer set contains more objects. Consequently, the energy consumed by retrieving result objects, which is far more than that used to retrieve index, becomes dominant. Hence, the difference between DSI and R-tree is reduced. For example, when $WSR = 0.02$, DSI only consumes 57.3% of R-tree's power consumption for window query under the REAL dataset, while the ratio of DSI's power consumption to that of R-tree reaches 94.9% when $WSR = 0.2$.

5.6 Error Resilience

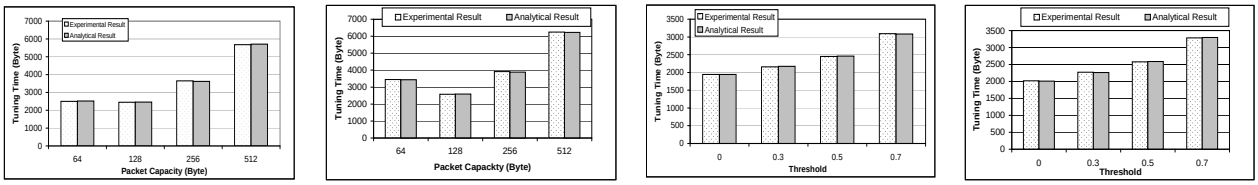
For a tree-structured index, *re-access* and *re-probe* are two straightforward strategies to resume a failed search. The former tries to access the lost packet from the next

[13] Note that the setup time typically refers to the time for a mobile device to start or to wake up its radio component.



(a) WQ ($T_{swi} = 0.1$ms) (b) WQ ($T_{swi} = 1$ms) (c) CWQ ($T_{swi} = 0.1$ms) (d) CWQ ($T_{swi} = 1$ms)

Fig. 26 Power Consumption of Window/Continuous Window Query (B=128kbps, C=128bytes, $WSR = 0.1$, $QLR = 0.1$)



(a) vs. C (θ=0.5,UNIF.) (b) vs. C (θ=0.5,REAL) (c) vs. θ ($C = 128$B,UNIF.) (d) vs. θ ($C = 128$B,REAL)

Fig. 27 Tuning Time Performance of Point Query

broadcast cycle, which incurs extremely long access latency. The latter re-starts the search from the packet right after the lost packet and hence all the previous efforts are in vain. Obviously, the performance of R-tree under an error-prone environment is not comparable to that of DSI by adopting either re-access or re-probe strategies. Every packet loss will incur an increase of the access latency and possibly the tuning time. In the literature, a progressive method has been proposed in [23] to efficiently resume an interrupted search by finding an available replicated index node. However, the solution is only applicable to the simple range query in one-dimensional space, and cannot be extended to answer complicated queries, such as kNN queries and CNN searches. Thus, we only evaluate the performance of DSI in this set of experiments.

To examine the resilience of on air spatial indexing techniques to wireless communication link errors, we compare their performance under an error-prone wireless broadcast environment, where a reliable data delivery is no longer guaranteed. We first evaluate the performance of point query under the default packet loss probability, i.e., ($\theta = 0.5$), with results depicted in Figure 27(a) and Figure 27(b). As we assume that the packets containing the final answers will not be lost, the access latency of point query will not be affected. It is observed that the results from simulation are pretty close to the analytical result. The difference of tuning time performance is only 0.1% and 0.2% under the UNIFORM and REAL datasets.

Next, we vary θ to evaluate DSI with various degrees of communication link errors. Figure 27(c) and Figure 27(d) show the performance of both datasets under different values of θ, which is varied from 0.0 to 0.7. As expected, simulation results match the analytical result. The difference is only around 0.6% and 0.2% for the UNIFORM and REAL datasets, respectively.

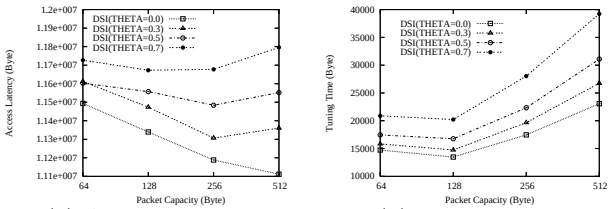
The performance of window queries and 30NN searches under different θ is also evaluated. Again, DSI provides a relatively stable performance. Table 4 summarizes the performance downgrade (in percentage) under various communication link error ratios for REAL dataset and that of UNIFORM dataset is ignored due to space limitation. Note that the baseline of comparison is the performance of DSI under the ideal, lossless wireless communication environment ($\theta = 0$). The loss of packets has a smaller impact on the performance of window queries, compared with that of kNN search. The reason is that the search range of window query is fixed, while the search range of kNN queries is determined based on the information obtained from the downloaded packets. When the packets cannot be retrieved successfully, the clients do not have sufficient knowledge to shrink the search range. As a result, it is likely to download irrelevant objects.

Capacity	θ	Window Query		30NN	
		Tuning	Latency	Tuning	Latency
64	0.3	0.05%	0.18%	8.41%	0.01%
	0.5	0.11%	0.01%	18.08%	0.0.03%
	0.7	0.27%	0.21%	36.20%	0.16%
128	0.3	0.03%	0.01%	8.83%	0.02%
	0.5	0.07%	0.01%	19.46%	0.01%
	0.7	0.17%	0.16%	40.67%	0.01%
256	0.3	0.06%	0.24%	9.66%	0.05%
	0.5	0.14%	0.01%	21.13%	0.06%
	0.7	0.31%	0.20%	42.36%	0.38%
512	0.3	0.12%	0.10%	10.79%	0.09%
	0.5	0.25%	0.38%	21.73%	0.23%
	0.7	0.55%	0.61%	43.64%	0.06%

Table 4 Performance Downgrade vs. θ under DSI (REAL)

In order to provide a visual representation of the performance under error-prone environments, Figure 28 presents the results of NN searches under the UNIFORM

dataset. It is obvious that the performance of DSI is relatively stable.



(a) Access Latency (b) Tuning Time

Fig. 28 Performance of NN Search under Error-Prone Environment (UNIFORM)

In summary, the fully distributed characteristics of DSI enable query processing to be resumed immediately after the loss of packets occurs. Distribution of index information to frames over the whole cycle contributes to the efficiency and robustness of DSI and the search algorithms in wireless data broadcast.

5.7 Multiple Data Types

In addition to the broadcast and query processing of homogeneous data type, we also conduct a set of experiments to evaluate the performance of the proposed sequential and integrated approaches for the broadcast of heterogeneous data types. We assume there are two uniform datasets with $N_1 = N_2 = 10000$, corresponding to two data types. An eight-bit string tag is allocated to represent data types. The performance of window queries with $WSR = 0.1$ is depicted in Figure 29, with respect to a window query on one data type ($D = 1$) and a window query on both data types ($D = 2$).

When the query is on a single data type, the sequential approach performs better than the integrated approach, requiring on average 91.3% of the tuning time and 92.3% of the access latency of the integrated approach. This is because the sequential approach uses a smaller index table, compared with that of the integrated approach. The number of index entries in an index table under the sequential approach is dependent on the number of frames inside the whole broadcast cycle, i.e., $n_i = \lceil log_2 \lfloor N_i/n_o \rfloor \rceil$. However, under the integrated approach, that number is bounded by the total number of frames inside one broadcast channel, i.e., $n_i = \lceil log_2 \lfloor N/n_o \rfloor \rceil$ with $N = \sum_{i=1}^{t} N_i$ and t being the number of data types supported in the system. Consequently, as the number of data types increases, the difference between the average index table sizes under the integrated approach and the sequential approach becomes more significant, which directly affects the performance.

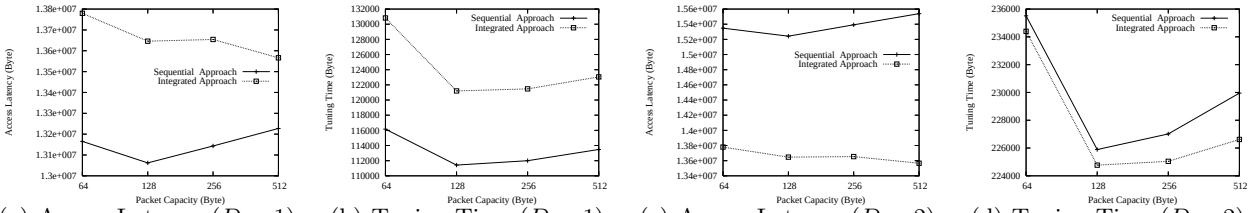
On the other hand, the integrated approach outperforms the sequential approach when the queries issued are on multiple data types, as depicted in Figure 29(c) and Figure 29(d). On average, it requires 99.1% of the tuning time, and 88.8% of the access latency of the sequential approach. The main reason is that the sequential approach forces the searches related to data objects from both data types to access both subcycles, while objects that are closely located are placed in nearby frames under the integrated approach.

Based on the performance of both approaches under different query types, we observe that the access latency of the integrated approach is stable while that of the sequential approach varies. The main reason is that the integrated approach treats objects of different types equally and objects are uniformly distributed along the wireless channel. Consequently, for a given query window, the average distance between the initial frame (i.e., the first frame retrieved after a query is issued) and the final target frame (i.e., the last frame retrieved that contains answer objects) is almost independent of the data types involved in the query. However, the sequential approach partitions the broadcast program into different sub-cycles. A query which involves objects of n data types has to scan n sub-cycles and hence n has a direct impact on the average access latency. In addition, we also observe that the tuning time performance of both approaches doubles when both data types are queried in one query. This is because when the number of data types involved into a query increases from one to two, the number of answer objects and hence the tuning time performance doubles.

In addition to snapshot window queries, we also evaluate the performance of the sequential approach and the integrated approach in answering continuous window queries, with both WSR and QLR set to 0.1. Again, two types of queries are evaluated, with one querying objects of one data type ($D = 1$) and the other querying objects from both types ($D = 2$). The performance is depicted in Figure 30. Similar observations as the result of the window query experiments are obtained. When objects of one data type are queried, the sequential approach performs better due to the small size of the index table. However, its advantages disappear when objects of multiple data types are queried in the same query. Due to the space limitation and the primary focus of this work, we are not able to elaborate the performance of two approaches in details in this paper. We plan to further examine this issue in our future research.

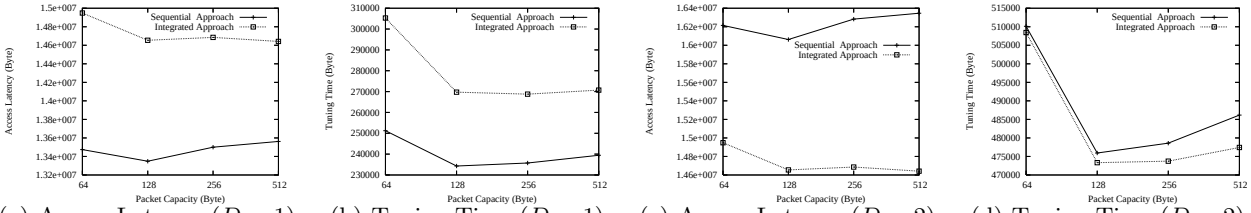
6 Conclusion

This paper addresses research issues in supporting spatial queries in wireless data broadcast systems. Spatial indexes proposed recently are all tree-based structures, in which a search starts at the root of the tree. A client searching for information on a broadcast channel has to wait for the data packet containing the root node to arrive, thus prolonging the initial probing time and the access latency. Moreover, existing studies ideally assume



(a) Access Latency ($D = 1$) (b) Tuning Time ($D = 1$) (c) Access Latency ($D = 2$) (d) Tuning Time ($D = 2$)

Fig. 29 Performance of Window Queries on one/two Data Types ($N_1 = N_2 = 10000$, $WSR = 0.1$)



(a) Access Latency ($D = 1$) (b) Tuning Time ($D = 1$) (c) Access Latency ($D = 2$) (d) Tuning Time ($D = 2$)

Fig. 30 Performance of Continuous Window Queries on one/two Data Types ($N_1 = N_2 = 10000$, $WSR = 0.1$, $QLR = 0.1$)

a reliable wireless communication environment, where packet loss never occurs. In fact, wireless communication is inherently unreliable. A desirable air indexing technique should be resilient in the error-prone wireless environment. That is, query processing should be quickly recovered after interruption. In this paper, a fully distributed spatial index, namely *DSI*, is proposed to meet these requirements.

DSI naturally combines multiple search paths into a linear structure and fully distributes this index structure over the whole broadcast cycle. As a result, it allows a search to start right after a client tunes into the channel. For the same reason, interrupted query processing can be resumed instantly even when a received packet is corrupted. Based on DSI, efficient search algorithms for energy efficient forwarding (EEF), snapshot window queries, snapshot k nearest neighbor queries, continuous window queries, and continuous nearest neighbor search are developed. An analytical model has been developed to measure both the access latency and the tuning time performance of EEF operation under both error-free and error-prone environments. An extensive simulation based evaluation is conducted under both error-free and error-prone environments. Experimental results show that DSI significantly outperforms a state-of-the-art spatial air indexing technique extended from R-tree.

In summary, the contribution of this work is five-fold:

- An error-resilient and energy-efficient spatial index structure, DSI, is developed for wireless data broadcast.
- Search algorithms based upon DSI for query points that are static or move on a projected trajectory are developed. These spatial queries include snapshot window queries, snapshot k-nearest-neighbor queries, continuous window queries, and continuous nearest-neighbor queries.
- An analytical cost model is derived for Energy-Efficient Forwarding operation.
- Two index organizations are proposed to support spatial queries on objects of multiple data types.
- An extensive simulation is conducted to compare the performance of DSI with a spatial air index extended from R-tree.

As for our next steps, we are considering multiple directions to extend DSI. First, we have focused on homogeneous data type in this study and proposed two approaches in this work to extend DSI for supporting spatial queries on objects of multiple data types. However, the issue of processing of spatial queries involving multiple data types deserves further investigation. We are working on a) efficient indexing and query processing algorithms to answer complex queries, and b) a general system framework which allows flexible performance tuning under different query patterns and object distribution patterns. Second, memory in modern mobile devices has become larger. Hence, caching data objects and/or index information on the clients is feasible. We are looking into techniques that combine caching and indexing to improve the query processing performance. Third, in our current work, we assume one Hilbert Curve which covers the entire service area. The service area might cover a large space, but a client is more likely to be interested in objects located close to her. Given a long broadcast cycle, it might take a long time for a client to reach the broadcast objects located close to her. This is an interesting issue that needs further investigation. We are currently working on a scalable solution. Last but not the least, we are looking into prototyping a wireless data broadcast testbed that supports spatial queries.

7 Acknowledgment

The authors would like to thank the editor Panos Kypros Chrysanthis, and anonymous reviewers for their valuable comments and suggestions that helped to improve the quality of this paper. In this research, Dik Lun Lee

was supported in part by a grant from the Research Grant Council, Hong Kong SAR, Chian under Grant no. 616005. Wang-Chien Lee and Ken C. K. Lee were supported in part by the National Science Foundation under Grant no. IIS-0328881 and IIS-0534343. Besides, Wang- Chien Lee was supported in part by the National Science Foundation under Grant no. CNS-0626709.

References

1. SA-1100 Microprocessor Product Brief. Website at http://bwrc.eecs.berkeley.edu/Research/Pico_Radio/Test_Bed/Hardware/Documentation/ARM/processor_arm_SA1100-productbrief.pdf.
2. D. Acharya and V. Kumar. Location based Indexing Scheme for DAYS. In *Proceedings of the 4th ACM International Workshop on Data engineering for Wireless and Mobile access (MobiDE'05)*, pages 17–24, June 2005.
3. 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 (SIGMOD'95)*, pages 199–210, May 1995.
4. M. Gruteser amd D. Grunwald. Anonymous Usage of Location-Based Services through Spatial and Temporal Cloaking. In *Proceedings of the 1st ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys'03)*, June 2003.
5. A.Seydim, M. Dunham, and V. Kumar. Location dependent query processing. In *Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'01)*, pages 47–53, Santa Barbara, CA, USA, May 2001.
6. M-S. Chen, K-L. Wu, and S. Yu. Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(1):161–173, January/February 2003.
7. A. Datta, A. Celik, J.K. 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 (ICDE'97)*, pages 124–133, April 1997.
8. 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.
9. Topologically Integrated Geographic Encoding and Referencing system. TIGER homepage. Website at http://www.census.gov/geo/www/tiger/.
10. N. Garg, V. Kumar, and M. Dunham. Information mapping and indexing in days. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, pages 951–955, 2003.
11. B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS'05)*, June 2005.
12. B. Gedik, A. Singh, and L. Liu. Energy Efficient Exact kNN Search in Wireless Broadcast Environments. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems (GIS'04)*, pages 137 – 146, 2004.
13. 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.
14. A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD'84)*, pages 47–54, June 1984.
15. 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'00)*, pages 157–166, February 2000.
16. T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficiency indexing on air. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 25–36, May 1994.
17. 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, March 1994.
18. 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.
19. Oliver Kasten. Energy Consumption. Website at http://www.inf.ethz.ch/personal/kasten/research/bathtub/energy_consumption.html.
20. 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), Special Issue on Database and Mobile Computing*, 4(3):205–227, July 1996.
21. W.-C. Lee and B. Zheng. DSI: A Fully Distributed Spatial Index for Wireless Data Broadcast. In *Proceedings of 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, June 2005.
22. 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, April 1997.
23. S-C. Lo and L-P. Chen. An Adaptive Access Method for Broadcast Data under an Error-Prone Mobile Environment. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 12(4):609–620, July 2000.
24. M. Mokbel, C-Y. Chow, and W. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'06)*, Seoul, Korea, 2006.
25. D. Moore. Hilbert curve. URL at http://www.caam.rice.edu/ dougm/twiddle/Hilbert.
26. Q. Ren and M. H. Dunham. Using semantic caching to manage location dependent data in mobile computing. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'00)*, pages 210–221, August 2000.
27. J.T. Robinson. The KDB Tree: A Search Structure for Large Multidimentional Dynamic Indexes. In *Proceedings of the ACM SIGMOD International Conference on*

Management of Data (SIGMOD'81), pages 10–18, Ann Arbor, MI, USA, April 1981.
28. H. Samet. The QuadTree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.
29. H. Schwetman. *CSIM User's Guide (version 18)*. Mesquite Software, Inc, http://www.mesquite.com, 1998.
30. A. Seydim, M. Dunham, and V. Kumar. An Architecture for Location Dependent Query Processing. In *Proceedings of the Fourth International Workshop on Mobility in Databases and Distributed Systems (MDDS'01)*, 2001.
31. J. Shanmugasundaram, A. Nithrakashyap, R. M. Sivasankaran, and K. Ramamritham. Efficient Concurrency Control for Broadcast Environments. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, pages 85–96, June 1999.
32. A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 422–432, April 1997.
33. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of conference of the Special Interest Group on Data Communication (SIGCOMM'01)*, pages 149–160, August 2001.
34. Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*, August 2002.
35. A. Wang, S. Cho, G. Sodini, and P. Chandrakasan. Energy efficient Modulation and MAC for Asymmetric RF Microsensor Systems. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, 2001.
36. J. Wang, R. Sinnarajah, T. Chen, Y. Wei, and E. Tiedemann. Broadcast and multicast services in cdma2000. *IEEE Communications Magazine*, 42(2), 2004.
37. 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.
38. 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)*, pages 239–250, March 2003.
39. J. Zhang and L. Gruenwald. Efficient Placement of Geographical Data Over Broadcast Channel for Spatial Range Query Under Quadratic Cost Model. In *Proceedings of the 3rd International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'03)*, 2003.
40. J. Zhang and L. Gruenwald. Optimizing Data Placement Over Wireless Broadcast Channel for Multi-Dimensional Range Query Processing. In *Proceedings of IEEE International Conference on Mobile Data Management (MDM'04)*, 2004.
41. B. Zheng and D. L. Lee. Information dissemination via wireless broadcast. *Communication of the ACM*, 48(5):105–110, 20056.
42. 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)*, pages 297–304, March 2003.
43. B. Zheng, J. Xu, W.-C. Lee, and D. L. Lee. Grid-Partition Index: A Hybrid Method for Nearest-Neighbor Queries in Wireless Location-Based Services. *VLDB Journal*, 15(1):21–39, 2006.