Efficient Information Retrieval in Mobile Peer-to-Peer Networks

Lijiang Chen[1] Bin Cui[1,2] Heng Tao Shen[3] Wei Lu[4] Xiaofang Zhou[3]
[1] School of EECS & Shenzhen Graudate School, Peking University {clj,bin.cui}@pku.edu.cn
[2] Key Laboratory of High Confidence Software Technologies (Ministry of Education), Peking University
[3] School of ITEE, The University of Queensland {shenht,zxf}@itee.uq.edu.au
[4] School of Information, Renmin University of China uqwlu@ruc.edu.cn

ABSTRACT

Mobile devices have become indispensable in daily life, and hence how to take advantage of these portable and powerful facilities to share resources and information begins to emerge as an interesting problem. In this paper, we investigate the problem of information retrieval in a mobile peer-to-peer network. The prevailing approach to information retrieval is to apply flooding methods because of its quick response and easy maintenance. Obviously, this kind of approach wastes a huge amount of communication bandwidth which greatly affects the availability of the network, and the battery power which significantly shortens the serving time of mobile devices in the network. To tackle this problem, we propose a novel approach by mimicking different human behaviors of social networks, which takes advantages of *Intelligence Accuracy* (IA) mechanism that evaluates the distance from a node to certain resources in the network. Extensive experimental results show the efficiency and effectiveness of our approach as well as its scalability in a volatile environment.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: [INFORMATION STORAGE AND RETRIEVAL], [Information Search and Retrieval], Search process

General Terms

Algorithms, Performance

Keywords

Information Retrieval, Mobile Networks, Peer-to-Peer

1. INTRODUCTION

Recently, with the advance in mobile wireless communication technology and the increasing number of mobile users, mobile devices, such as cell phones, PDAs and pocket PCs, have become more and more popular in daily life. These mobile devices generally have gigabytes of storage capacity and can connect to each other using short-range communication access, such as Bluetooth [1] and IEEE802.11 [2]. With such portable devices in hand, we can bring hundreds of documents and multimedia files (i.e., mp3 songs or short movies) wherever we go. Given these facilities, we can imagine a scenario where different people are interconnected in a confined area. For example, in a business district with adjacent buildings, people can use their mobile devices to search and share information and resources (e.g., mp3 music files, e-books, working documents and advertisements), or to find a person providing a particular service. This kind of environment is very popular in real life, such as in schools, business centers, shopping malls, and airports, etc., where mobile nodes can interconnect with each other in a peer-to-peer (P2P) way to form a mobile P2P network.

In such a mobile network, all nodes are generally gathered together to form a P2P network and free to move, join or leave. Since the nodes in the network are highly dynamic or periodically participate, it is very expensive to maintain a structured network overlay. Therefore, a reasonable and practical solution is to construct the nodes in an arbitrary unstructured way [7, 9]. In this manner, nodes just dynamically maintain neighbors in their reachable ranges like real neighborhood in social networks, and no special overlay is needed. Specifically, mobile P2P environment differs from the traditional P2P systems based on static nodes. First, nodes in P2P networks are PCs connected through internet. They have powerful computing capacity, large storage and stable interconnection. The situation is totally different in mobile P2P environment, as mobile devices have limited resources (computing capacity, storage, bandwidth, battery power), which means high resource consuming algorithms used in P2P networks are not suitable for mobile P2P. Second, nodes in mobile P2P networks are connected via short-range communication access (e.g. IEEE802.11). Thus neighbors are adjacent in physical network and data is transferred in multi-hop manner between non-neighbor nodes. However, two P2P nodes can directly communicate via IP address. Third, mobile nodes move from time to time, which makes the network topology and neighbor relationship vary rapidly. Thus, information frequently used to process queries in P2P literature, e.g. profile of neighbors and query historical records, can not be directly applied. In a word, query forwarding in mobile P2P environment suffers from more limitation, uncertainty and complexness.

CIKM'09, November 2–6, 2009, Hong Kong, China.

Most previous works in mobile P2P networks [4, 8, 13, 23, 17, 24] focus on data dissemination and resource discovery problems, and mainly study how to push the information to end users in the network. In that case, users are passive receivers of information. However, most of the users are likely active searchers of information in real-life applications, where the users can seek resources they want, search specific information for certain purposes, find discount advertisements, or download music or files from the network. In this paper, we aim at providing active search for end users. Thus, we tackle the problem of keyword-based query forwarding in file-sharing unstructured mobile P2P networks. Query forwarding differs from message routing in mobile P2P environment. In message routing, the destination of a message is known. Nodes have a prior knowledge of the routing paths. However, in query forwarding, there is no prior knowledge of where to send the messages. This unknown destination implies more uncertainty and less control in message relaying. Therefore, the algorithms of distance-aware routing are not appropriate for query forwarding problem.

There exist some challenges for efficient search in such mobile P2P networks. First, the query routing strategy must be efficient although the global index is not available. Second, due to the constraint of computation capacity, battery energy and communication bandwidth, the query forwarding algorithm must be light-weight and energy-efficient. Third, since the network is highly vulnerable, the solution must be able to adapt to frequent data updates and high network fluctuation.

To deal with the above challenges, we adopt the human strategies in social networking for mobile P2P information retrieval problem. Differing from previous works [18, 12] in social networks which utilize social-links, interesting-similarity or behaviors-alike properties, we exploit the human natural behaviors of finding answers through acquaintances, learning from experiences and gossiping words. Intuitively, once a person has a question, he would ask an acquaintance who has that knowledge, or who has a wide knowledge that can potentially answer his question. In our proposed solution for mobile P2P networks, we use intelligence information about neighbor peers instead of knowledge of acquaintance. The intelligence information is a piece of message about where to find a specific data (file). We use Intelligence Accuracy (IA) to evaluate the precise of all these intelligence information provided by its neighbors, which helps peers to find a better node to further pass the query to its neighbors. In our implementation, the Intelligence Accuracy is a convenient yet efficient method to evaluate the distance from a query node to the resource provider, with which the shortest way can be determined for the whole search operation in a volatile environment without a global view. The intelligence information is exchanged with neighbor peers as the way social peers gossip words, and spread in neighborhood. It helps peers collect more information from their neighbors. What's more, the peers learn from their query experience, both in success and failure. After that, they re-adjust the Intelligence Accuracy (IA) value of the intelligence they used, and make it more precise and up-to-date. With above mimicking mechanisms from social networks, we propose an intelligence-based approach for information retrieval and demonstrate that it can well tackle the challenges and provide efficient and effective information retrieval in mobile P2P networks. We further discuss our proposed approaches under the circumstances of network fluctuation and data updates to show its scalability and maintainability.

In summary, our approach is such a keyword-based information retrieval algorithm that is light-weight, energy-saving, search-efficient and self-adapted, which is well-fit for the mobile P2P environment. Our contribution includes:

- We propose a novel approach for efficient keyword search in file-sharing mobile P2P networks, based on the mechanisms mimicking natural human behaviors of finding answers through acquaintances, learning from experiences and gossiping words in social networks.
- We take advantages of the Intelligence Accuracy (IA) mechanism, which keeps track of intelligence information about the timely distance of files in the network, to improve the performance of the queries.
- We specify the processes of our proposed approach in the situation of network volatility and data updates, together with cost models to analyze the energy consumption and communication cost to give a deep view of the performance of our approach. We show that our approach is functionally self-adaptive in highly dynamic networks.
- We conduct extensive experiments to evaluate the performance of our proposed approach. The results show that it is a time efficient and highly self-adaptive approach with balanced query load and high success rate.

The rest of the paper is organized as follows. In the next section, we review the related work. We present the details of our intelligence-based information retrieval approach in Section 3. Section 4 describes a performance study and gives a detailed analysis of results. We finally conclude the paper in Section 5.

2. RELATED WORK

In mobile P2P networks or wireless ad hoc networks, a number of research studies have been conducted on data dissemination and discovery [6, 14]. Siddhartha et al. [20] proposed a P2P data dissemination protocol over an ad hoc network. It focuses on data transmission, in which a file is divided into segments according to a particular mobility model, while it ignores the issue of data search. Papadopouli et al. [17] introduced a peer-to-peer architecture called 7DS, which enables data sharing in a self-organizing fashion with either mobile or stationary nodes by simply using random forward method. Das et al. [8] studied a cooperative downloading strategy for content delivery and sharing in wireless vehicular-to-vehicular (V2V) networks, called SPAWN. It leverages the inherent broadcast nature of the wireless medium for neighbor discovery and uses a proximity-based piece-selection strategy to exchange data pieces. Klemm et al. [11] presented a special-purpose system for search and file transfer tailored to both the characteristics of MANET and the requirements of peer-to-peer file sharing. Motani et al. [15] proposed a wireless virtual network called PeopleNet, which mimics the way people seek information via social networking. It divides nodes into groups and propagates queries to a particular group based on given types. Wolfson et al. [22] proposed a rank-based dissemination (RANDI) algorithm which considers three constraints of mobile devices,

including energy, communication bandwidth and storage. Tchakarov et al. [21] proposed a Geography-based Content Location Protocol (GCLP) to make use of physical location information for efficient content location in ad hoc networks. Lupu et al. [14] introduced a fast data dissemination method for structured P2P networks with mobile devices. It quickly constructs a CAN-like [19] structured P2P network for people who stay together with a short period of time. Most of above algorithms investigate how to disseminate the data in a peer-to-peer environment, which differ from our work that they push the data to the end users while we focus on processing the user-issued queries.

More recently, Fiore et al. [10] proposed an algorithm called Eureka to retrieve user contents in MANET. To the best of our knowledge, Eureka is the most related work to our research. It focuses on the query propagation and applies a pure peer-to-peer approach to conduct data retrieval. Mobile nodes with Eureka estimate the information density in their proximity and use it to direct queries toward areas where the requested data is denser. However, it suffers from high maintenance cost of indexing information density estimation of all data items in every node. It also suffers from high communication overhead caused by information updating because a single change of information density estimation can incur the updates of all relevant nodes in the network. Our proposed IA approach assumes a more general environment with users in different mobilities, and it is well adapted to the highly dynamic environment with frequent data/peer update.

3. INTELLIGENCE-BASED INFORMATION RETRIEVAL

In this section, we introduce our intelligence-based approach for information retrieval in mobile P2P networks, which mimics social human behaviors to facilitate efficient information retrieval. It re-adjusts IA values after query processes just like people learning from their experiences; it updates and exchanges the intelligence information as the way people gossip; its strategies to forward queries like people finding answers from acquaintances.

As shown in Figure 1, a query routing process can be divided into several forwarding steps. At each step, the participant forwards the query to one of its neighbors within its reachable range, by checking the intelligence information provided by them. Here, we define a node's reachable range as the range in which the node can reach all other nodes in one-hop connection. We use query *time-to-live* (TTL) to control the hops of query propagation. If the query routing path's length is equal to TTL and the result is still not found, the query fails in finding its results. Otherwise, the resource provider node (denoted as *R-provider*) will send the result back along the query routing path to the query issuer node (denoted as *Q-issuer*). As discussed in [25, 16], an appropriate value of TTL is needed to tradeoff the success probability of queries and the overall query load in the network. However, this value is highly dependent on the workload and the network scenario.

3.1 Intelligence Accuracy

Before discussing the query forwarding strategy, we introduce the notion of *intelligence* and our method to measure the accuracy of intelligence information. In this paper,

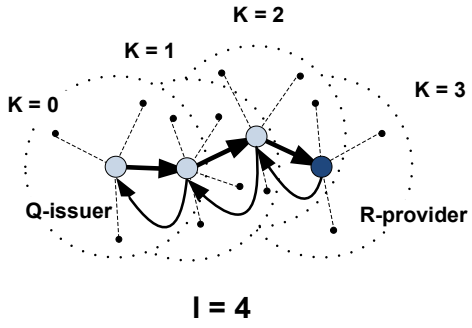


Figure 1: An Example of Query Routing

the intelligence of a query item in a node is the information about the distance from the node to the result item. This kind of intelligence is collected from neighbors' gossip-like intelligence exchanging and query forwarding experiences. They are stored in the index tables of each node, as shown in Figure 2. The index table at left side, named *self-contained items* (SCI) table, stores the intelligence of data items which are provided by the node itself, obtained from previous queries or evaluated from *neighbors' items* (NI) table. This information is the node's own intelligence which will be sent to its neighbors for exchanging. The NI table at right side maintains all selected intelligence collected from neighbors which can help to decide where to forward a query.

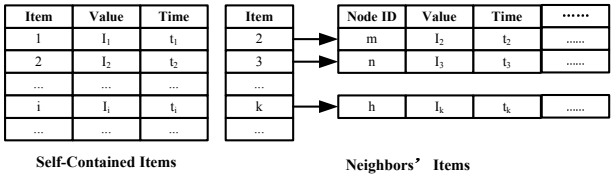


Figure 2: The Indexing Structures

The *Intelligence Accuracy* (IA) is a criterion representing the evaluating distance from the nearest copy of the resource data. We denote data item j's Intelligence Accuracy value as I_j. Additionally, since the mobile P2P network is a highly dynamic environment, we bring in a time parameter t_j to indicate the timeliness of intelligence. We measure IA in following ways:

- For the self-provided data items of a node, we assign them a fixed value I^0 and a fixed time parameter t^0. I^0 is the maximum value of IA in the network, while t^0 denotes that the data is the original copy and is always valid unless it is explicitly deleted or updated. They are stored in the SCI index table as shown in Figure 2. Whenever we find I^0 in the NI table, we are sure that this intelligence is accurate and we can get the resource at next hop.

- After forwarding each query, the IA value of the query item will be re-adjusted and the time parameter will be updated, which exploits the experiences from query processes for refinement. This process makes the intelligence information up-to-date. The re-adjustment of the IA value follows the same principle for the data item, i.e., if the node is nearer to the resource provider,

its IA value is higher. After a query request is finished, the corresponding result is cached in the query issuer for a certain period of time. This newly cached data item will undoubtedly affect the IA value of neighbor nodes around it, which should be taken into consideration during adjustment. We re-adjust the IA value in a node according to its position between the node with cache and the resource provider. As illustrated in Figure 3, the nodes closer to either of those two nodes have higher IA values and the node in the middle of the path has the lowest IA value because it is the one who is farthest to both data copies.

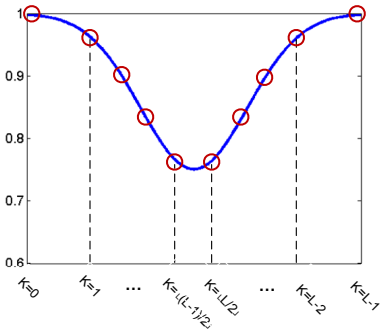


Figure 3: The Illustration of IA Re-adjusting

In order to achieve our IA value re-adjusting plan, we use an adjusted normal distribution function for new value calculation illustrated in Figure 3.[1] Referring to the example in Figure 1, we assume that the query item is item j, node i is one of the forwarding nodes in j's query forwarding path, i's position is k and the length of that path is l. We use the following formula to re-adjust IA value of j:

$$I_j = \begin{cases} I^0 - \frac{1}{\frac{1}{l}\sqrt{2\pi}} e^{-\frac{(k-\lceil\frac{l+1}{2}\rceil)^2}{2(\frac{1}{l})^2}} I^0, & l < TTL \ (1) \\ 0, & l = TTL \ (1') \end{cases}$$

Formula 1 is an adjusted normal distribution function with a mathematical expectation parameter $\mu = 1/l$ and a standard dispersion parameter $\sigma = \lceil (l+1)/2 \rceil$. Take the query forwarding process shown in Figure 1 as the example. Node $k = 0$ and node $k = 3$ have the same IA value I^0. While node $k = 1$ and node $k = 2$ have the same IA value $I < I^0$, since they both have one-hop distance to the data item. We specify that if there are two duplicate items within the reachable range of a node, that node will only choose one of them randomly. If the query fails, all the nodes in this query path set their corresponding IA value to 0, because their intelligence information is out-of-date. With this formula, we can also re-adjust the IA value without the effect of the query path length l, which makes the IA value more precise. Suppose that there exist two nodes u and u', they are both in the second position ($k_u = 1$ and $k_{u'} = 1$) of their own query path l_u and $l_{u'}$, where $l_u < l_{u'}$. Though their position are the same, the intelligence information they provided are not the same at beginning. But after re-adjusted, they should have the same IA value because they both have one-hop distance with the new cached result. The formula can be automatically adapted to guarantee the correctness of that situation.

[1]In case that L is odd number, the two circles in the middle actually represents the single trough point of the curve as $\llcorner L/2 \lrcorner = \llcorner (L-1)/2 \lrcorner$.

- Each node collects intelligence from its neighbors with gossips and updates its NI table, using the following formula:

$$I_j = \{ I_{i,j} \mid 1 \leqslant i \leqslant n_j \wedge \mathcal{F}_{i,j} = \mathcal{F}_{max} \} \quad (2)$$

$$\mathcal{F}_{max} = max_{1 \leqslant i \leqslant n_j} \{ \frac{I_{i,j}}{(\triangle t_{i,j})^2} \} \quad (3)$$

where $I_{i,j}$ is the IA value of query item j in neighbor i and $\triangle t_{i,j}$ is the corresponding time interval between current time and the latest update time of $I_{i,j}$. n_j is the number of neighbors of node j. $\mathcal{F}_{i,j}$ is the forwarding weight of $I_{i,j}$ and $\mathcal{F}_{max}$ is the one who has the maximum forwarding weight. The forwarding weight $\mathcal{F}$ is a criterion to tradeoff the timeliness and IA value. Here, we store the item $I_{i,j}$ with the maximum forwarding weight from all neighbors in every update process. Since the NI table is directly devoted to our intelligence-based forwarding strategy, the additional maintenance of NI table entries will be discussed in Section 3.2.

- At the same time, the nodes periodically update their SCI tables after their NI tables have changed, using the following formula:

$$I_j = \frac{I'_j + I''_j}{2} \quad (4)$$

where I_j represents item j's new IA value in SCI table, I'_j and I''_j represent j's former IA values in SCI and NI tables respectively. After that, they send the intelligence of all the updated items to their neighbors, whose values are larger than a threshold δ which will be discussed in Section 3.2. Note that, in a mobile P2P network, neighbors of a node are always changing. Hence, indexes of NI table are changed frequently. However, we integrate the intelligence from NI table into SCI table periodically, for the sake of reducing the communication cost incurred by frequent updates.

In our implementation, to expedite the data dissemination in mobile environments, we store the result in the query issuer node for a certain period of time. During this period, the query issuer node can serve this resource to the others. If the data storage in the query issuer node exceeds a limitation, it forces the oldest cache to be deleted. This duration for cache can be a system tuning factor and may vary for different applications. The duration of cache is generally short and during this period we consider the cache of resource to be valid using a version number to distinguish it from new update versions (refer to Section 3.3). We don't cache results in intermediate peers because it is a tradeoff between data dissemination and device storage consumption.

3.2 Intelligence-based Forwarding Strategy

We can see that, Intelligence Accuracy is a value reflecting the distance from a node to the demanded resource. For an

item j, a node whose IA value is nearer to I_j^0 is closer to one of item j's providers in the network. Therefore, the node forwards the query to one of its neighbors who has the highest value of I_j. However, if we take the time factor into consideration, the IA value updated more recently is expected to be more precise. Integrating the above considerations, we design our query forwarding strategy as follows:

- Given a query request, if node i itself has the result, it forwards the result back to the requested neighbor.
- If the result is not found in node i, it forwards the request to one of its neighbors $\mathcal{N}_k$ who has the maximum forwarding weight $\mathcal{F}_j$ of item j, which is recorded in its NI table according to Formula (2) and (3):

$$\mathcal{N}_k = \{\mathcal{N}_{i,k} \mid 1 \leqslant k \leqslant n_i \wedge I_{(\mathcal{N}_{i,k}),j} \in NI_i\} \quad (5)$$

  where n_i is the number of neighbors of node i, $I_{(\mathcal{N}_{i,k}),j}$ is the item provided by neighbor $\mathcal{N}_{i,k}$ and NI_i is the NI table of node i.
- If the values of I_j in all neighbors are 0, i.e., no neighbor has intelligence about the query item j, node i forwards the query to the neighbor according to following formula:

$$\mathcal{V}_{max} = max_{1 \leqslant k \leqslant n_i}\{\frac{\sum_{I_i \in SCI_k} I_i}{|SCI_k|}\} \quad (6)$$

  where SCI_k denotes the SCI table of the k^{th} neighbor out of its n_i neighbors and $|SCI_k|$ is the number of entries of SCI_k. $\mathcal{V}_{max}$ denotes the neighbor who has most accurate intelligence information. node i forwards the query to the neighbor with $\mathcal{V}_{max}$, which is like people asking questions to an acquaintance with a wide knowledge. The node with more accurate intelligence information indicates that more query requests can find their results through it. Therefore, the probability of finding a query result through this node is higher. The $\mathcal{V}_{max}$ is calculated by the neighbors themselves and exchanged along with update messages.

Intuitively, in order to satisfy the forwarding strategy, a node must store the indexes of every item from every neighbor, which is infeasible in reality. Therefore, in our implementation, we simplify the indexes by exploiting the strategy on NI table's updating phase. As shown in Figure 2, we manage the indexes in NI table based on items. For one item j, we store one entry which has the highest forwarding weight $\mathcal{F}_j$ (Formula (2) and (3)). When a new updated value from a neighbor comes, we compare it with the stored value. If the new $\mathcal{F}$ is higher, we update the entry. Otherwise, we drop the update message. Nevertheless, we still have to keep the indexes of all items in our index tables. However, we discover that if an IA value is so small as that its evaluating distance from the resource is longer than the TTL, this IA value becomes useless. Thus, we can remove it from our index tables. For the sake of index optimum, we set a threshold δ:

$$\delta = 2^{-TTL} \quad (7)$$

The IA values in both SCI and NI tables which are smaller than δ can be safely removed.

One problem of the approach we presented above is that the query forwarding may get locked into a cycle. For instance, node i forwards the query to node j, j then forwards it to node k. And the IA value in node k's NI table points to node i, so k forwards it to i. This cycle will recur until the TTL is reached. The query forwarding fails to explore other parts of the network where the results may reside. To solve this problem, we enhance the strategy with an additional rule that a new IA value is recorded in a node's NI table only if this new value is larger than the node's own IA value in SCI table. With this rule, we guarantee that a node always forwards a query to a neighbor with a larger IA value. In case that such a neighbor does not exist, it randomly selects a neighbor to forward. Furthermore, in order to improve the robustness of our algorithm, we can store entries with top-k $\mathcal{F}$ of one item in our NI table by any chance the best candidate neighbor is disconnected at the moment we need it.

3.3 Network Dynamic and Data Update

The mobile P2P network is such a dynamic network that mobile devices can join or leave freely. Nodes in this network can move from one place to another, and can add, delete or modify data arbitrarily. As we know, communication cost and maintenance cost can be extremely high in this kind of volatile environment. We show that our IA approach does not need any additional accesses to deal with the volatility. We introduce how our IA approach tackles the network dynamic and data update as follows:

- **Node joining or leaving**. If a new node joins the network, it connects to neighbor nodes in its reachable range by sending its IA value of all self-contained data. The neighbor nodes receive the message and treat it as a normal NI table updating process. When a node leaves the network, it can either get away without doing anything or notify all its neighbors. Even though a node leaves in silence, all intelligence information from it will die out soon or later by the actions of query forwarding and NI updating.
- **Data insertion or deletion**. For data items of a node, if a data is deleted, it sets the data's I and t to 0 in SCI table and notifies its neighbors. If a new data is added, it adds a new entry into SCI table, sets the value of that data to be I^0 and t^0, and notifies its neighbors. The notification uses NI update process and is sent in the periodical update time slot. The neighbor nodes who receive that kind of notification can just execute a normal NI table updating process.
- **Data modification**. If the data is modified, it is practically difficult to update it in-place throughout the network because of the high network volatility and widespread replicas. As an alternative, we install new versions of the data. Thus, the modified data is inserted into the network as new data. Query nodes can distinguish new data from old ones according to their version number. A query node always prefers a newest version of the data among all ones it can find. Even though this found version may not be the newest one in the network, we consider it to be valid before newer once can be found. If a node discovers a newer version of data, it will discard the index value and cache replica of the old one and only change its index value to the new one.

It is clear that no additional process is needed to deal with the network volatility and data update, besides NI updates. In other words, our model is a self-adaptive model who can be well fitted to highly dynamic networks.

3.4 Analysis of Cost Model

3.4.1 Communication Cost

We define the communication cost C as the amount of messages which have been sent, including query request messages, result messages and update informing messages. For clarity of presentation, we simply denote the size of a single message as m. Note that, we can set different sizes for different types of messages. We now investigate the process of query routing and IA value updating to analyze the cost of communication.

As mentioned before, the length of query path is l if the result is found. Otherwise the number of routing hops is equal to TTL. During the query routing, each node chooses one of its neighbors by checking its own index, then forwards the query request to that neighbor, which sends only one message. So does the result returning process. Therefore, the communication cost C is

$$C = 2 \times (l-1) \times m, \quad l \leq TTL \tag{8}$$

The IA value updating process is one of the key processes in our model, because many processes are essentially IA value updating processes, such as node joining/leaving process, data adding/deleting process and data modifying process. For a node i, we assume that it has n_i neighbors. In the updating process, node i sends only one message m_i to each of n_i neighbors. After receiving the message, those n_i neighbors just update their own NI table and nothing more need to be done. The communication cost of node i's IA value renewing is

$$C = m \times n_i \tag{9}$$

3.4.2 Energy Consumption

We consider the energy consumption of battery while nodes are forwarding the queries or updating IA values. Suppose e_i is the energy cost of node i for one-hop connection. As we know a mobile device can be configured with several power levels at which it can send packets. Different power levels lead to different reachable ranges. To simplify our model, we let p to be a standard power level in our network. Each node sends messages at power level p and consequently has the same reachable range. In general, the energy cost e_i of sending a data unit by node i is

$$e_i = p \times \sigma_i \tag{10}$$

where σ_i is a battery state parameter. Here σ_i reflects node i's internal states such as remaining electric energy of the battery and the valuation of each unit of power.

We distinguish two cases in explaining the energy cost during query routing:

- If node i is not on the query routing path, then $e_i = 0$.
- If node i is on the query routing path, then whenever node i forwards a data unit of query request or query result at power level p, the corresponding cost e_i is $p\sigma_i$.

On the other hand, the receiving energy cost of a node i for a data unit is a fixed value r_i. We now consider energy cost of different roles in the network. For an arbitrary intermediate node E_{itm}, it receives and forwards the message twice in both query propagation process and result delivery process. Therefore, the energy cost is

$$E_{itm} = 2(p\sigma_i + r_i) \times m \tag{11}$$

For an issuer node E_{isr} or a resource provider node E_{rsp}, it only receives and forwards the message once. Thus, the energy cost is

$$E_{isr} = E_{rsp} = (p\sigma_i + r_i) \times m \tag{12}$$

For a node who updates its IA value to its n_i neighbors. The energy cost E_{upd} is

$$E_{upd} = p\sigma_i \times m \times n_i \tag{13}$$

3.5 The Query Processing Algorithm

In this subsection, we introduce the overall query processing algorithm of our IA approach as shown in Algorithm 1. When a node i receives a message M from one of its neighbors (line 2), it first checks the type of this message. If it is a query request message (line 3), it checks whether the query has been processed. If so, it drops the message (lines 5-6). If query TTL is equal to zero, the query fails. A failed message is returned (lines 8-9). Otherwise, it searches its own data for the query (line 11). If the result data is found, node i calculates the length of query path l, and then sends the result data, l and its position k back to the previous hop neighbor (lines 12-14). On the contrary, if the result is not found, it calculates its own position k in the query path and adds the query into its query list in order to record which one it has already processed and where it is from. After that, it checks for next hop (lines 16-19). If an appropriate neighbor is found, node i forwards the query to it, or else forwards the query to the neighbor who has maximum forwarding weight $\mathcal{V}_{max}$ (lines 20-23). If node i receives a result data message (line 28), it first checks whether this query was issued by itself. If so and the query is successful, it stores the data and updates the corresponding IA value. If the query fails, it re-issues the query with a pre-defined number of attempts (lines 30-35). Otherwise, it searches the query list to find the record of that query (line 37). If no record is found, the query has been sent to the wrong place, node i drops it (line 44). If a record is found, node i updates the corresponding IA value and then sends it to the next hop. Otherwise, it directly forwards to the next hop (lines 38-42).

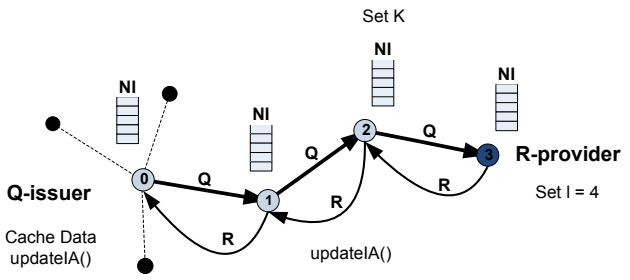


Figure 4: The Illustration of Query Process

Algorithm 1 The Query Processing Algorithm

```
1: k = 0; l = 0;
2: while recvMsg(M) do
3:   if M.type == QUERY then
4:     Q = M.query;
5:     if hasProcessed(M.qID) then
6:       Drop(M);
7:     end if
8:     if Q.ttl == 0 then
9:       return (Data = Failed);
10:    else
11:      Data = checkData(Q.dataID);
12:      if Data! = NULL then
13:        l = TTL − Q.ttl + 1;
14:        return (Data, l, l − 1);
15:      else
16:        k = TTL − Q.ttl;
17:        addQuery(Q, k, fromID);
18:        Q.ttl = Q.ttl − 1;
19:        nextHop = searchNI(Q.dataID);
20:        if nextHop! = NULL then
21:          sendTo(nextHop, Q);
22:        else
23:          sendTo(V_max Nb, Q);
24:        end if
25:      end if
26:    end if
27:  end if
28:  if M.type == RESULT then
29:    Q = M.query;
30:    if Q.issuer == my.ID then
31:      if M.Data! = Failed then
32:        addData(M.Data);
33:        updateIA(M.dataID);
34:      end if
35:      Re − issueQuery(Q, try − 1);
36:    else
37:      Q = searchQueryList(M.qID);
38:      if Q! = NULL then
39:        if M.Data! = Failed then
40:          updateIA(M.dataID);
41:        end if
42:        sendTo(Q.nextHop, M.Data);
43:      else
44:        Drop(M);
45:      end if
46:    end if
47:  end if
48: end while
```

EXAMPLE 1. *Figure 4 shows a detailed example of how query processing uses intelligence-based forwarding strategy in the mobile P2P network. Node* 0 *is a query issuer. It issues a query* Q*. It then lookups its NI index table to find a neighbor to forward that query. It decides to send the query to node* 1 *according to the intelligence-based forwarding strategy. Node* 1 *receives the query message from node* 0*. It finds out that it has no data for that query. Then it calculates its own position* k *in the query path and forwards the query to node* 2 *after searching its NI table. Node* 2 *conducts the same job as node* 1 *does and forwards the query to node* 3*. Node* 3 *checks its data and finds out it has the result data. It then calculates the length* l *of the whole query path and then sends the result back with the value of* l*. Node* 2 *and node* 1 *receive the result data, update their corresponding IA value and send back to the next hop retrieved from the query record. Finally, node* 0 *receives the result, and updates the IA value.*

4. PERFORMANCE EVALUATION

4.1 Simulation Settings

In this section, we evaluate our approach as well as two comparative approaches, Naive approach and Eureka [10], in the network simulator ns-2 [3]. The naive approach simply disseminates the queries in a flooding way. We use it as a baseline approach because it is simplest in maintenance yet proved to be efficient in search though with obvious drawbacks. We also compare our algorithm with Eureka because the Eureka approach is the most recently proposed approach related to our work and has been proved to be more effective than several previous works. In the Eureka approach, each node maintains information density evaluation of resources in the surrounding range, within which it can reach other nodes in multi-hops. This range is called a region. Queries are propagated to a region that is evaluated to have higher information density.

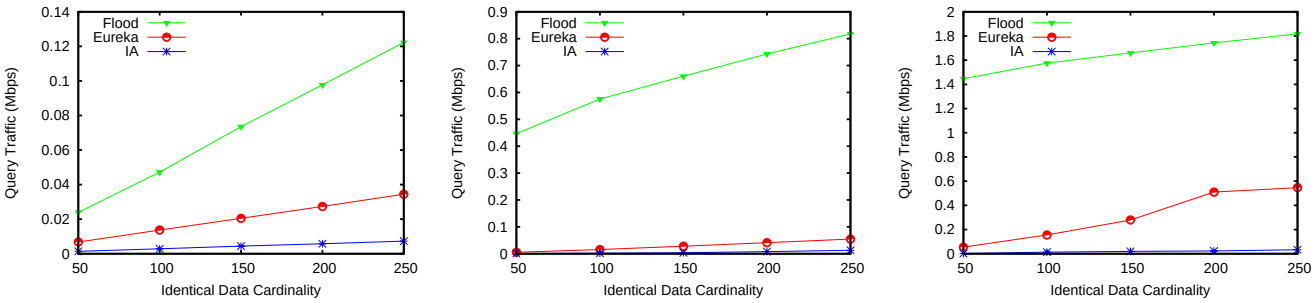
4.1.1 Mobility Model

In the simulation, we set the scenario of a $1000m \times 1000m$ area. Within this area, the mobile nodes move according to the Random Waypoint Mobility Model [5]. In particular, at the beginning of the simulation N_s percentage of all N mobile nodes are randomly placed in this area. Each node has a destination which is another random point within the space. The node moves to its destination at a constant speed s along the straight line. After it arrives, it pauses at the destination for a period t_s, then randomly chooses another destination, and repeats this procedure. N_d ($N_s + N_d = 1$) percentage of nodes are randomly placed outside the area. Each of them moves toward a random destination within the area at speed s and pauses at the destination for t_d. After that, it moves toward outside of the area and is removed from the simulation. The pause time t_s is uniformly distributed between 120 and 240 seconds for stable nodes. t_d is uniformly distributed between 30 and 60 seconds for highly dynamic nodes. The motion speed s is randomly selected from the interval $[0.5, 1.5]$ miles/hour for modeling the pedestrian scenario. This mobility model simulates, for example, the motion of workers and visitors at business district. Within the area, there are D identical data resources and each of them has two replicas randomly distributed among nodes. Every node selects a random data for issuing a query. The query rate is γ. All mobile nodes use IEEE 802.11 for connection. The bandwidth is 10 Mbps and the transmission range is 100 meters. The parameters used are listed in Table 1.

4.1.2 Performance Metrics

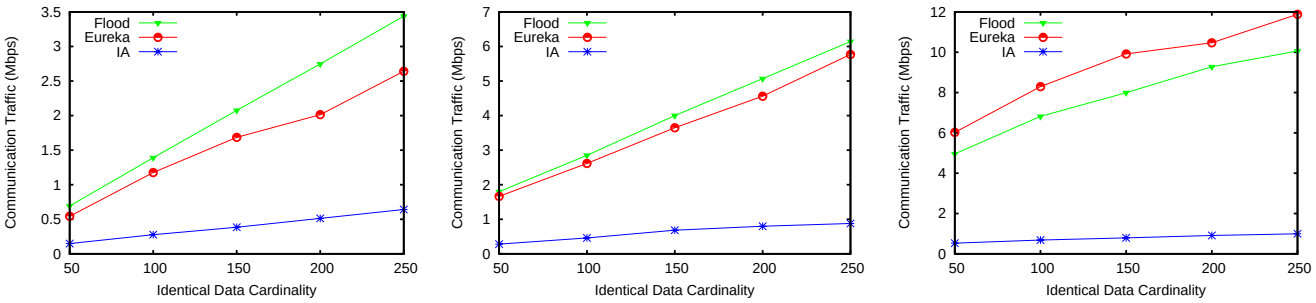
We exploit the following performance metrics:

- **Query Traffic**: the amount of traffic that is incurred by the query propagation in the whole network. It is clear that query traffic is affected by the query rate γ. But, in this experiment, we only compare the traffic



(a) 100 Nodes (b) 500 Nodes (c) 1000 Nodes

Figure 5: Performance of Query Traffic



(a) 100 Nodes (b) 500 Nodes (c) 1000 Nodes

Figure 6: Performance of Communication Traffic

Table 1: Parameter Setting

Name	Value
Simulation Time	2000 seconds
D	$50, 100, 150, 200, 250$
N	$100, 500, 1000$
N_s	80%
c	1.0
p_b	0.1
γ	0.2/sec
λ	0.035/sec
δ	0.05

of different approaches with the same γ, because the interval trends of those approaches are similar with different γ.

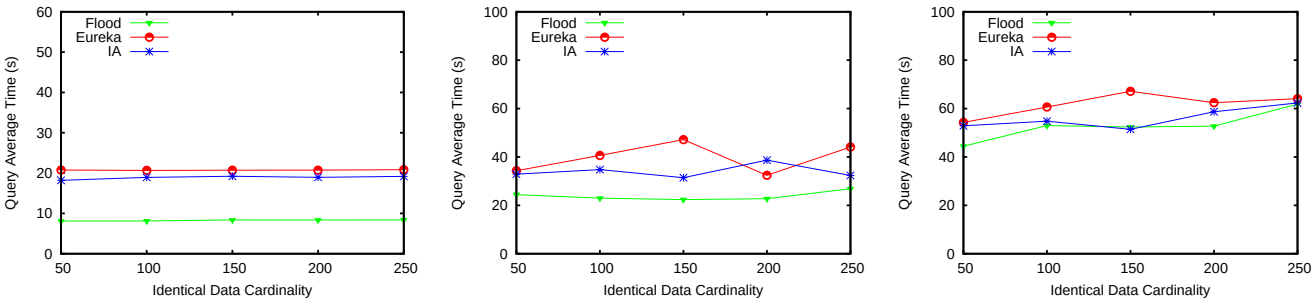
- **Communication Traffic**: the total number of messages that are transferred during the whole simulation. In our approach, we compute this cost according to the model discussed in Section 3.4.1 and we count only the messages produced in the application layer. It reflects the whole network traffic. If an approach incurs too much communication traffic, it wastes more bandwidth.
- **Query Response Time**: the overall duration from query execution, i.e., from the time that a query is issued until the result is received. Response time is one of the important criterions of query efficiency. We aim at shortening the response time to improve the performance.
- **Query Success Rate**: the percentage of queries that have eventually received the required resource data. It is denoted as

$$Q_{SR} = \frac{\Sigma|success\_queries|}{\Sigma|all\_queries|}$$

- **Energy Consumption**: we measure the energy consumption of a node while it participates in the whole query routing process. We follow the aforementioned model proposed in Section 3.4.2. In reality, users are very sensitive to the energy consumption of the battery, because it imposes a great impact on the using time of mobile devices.

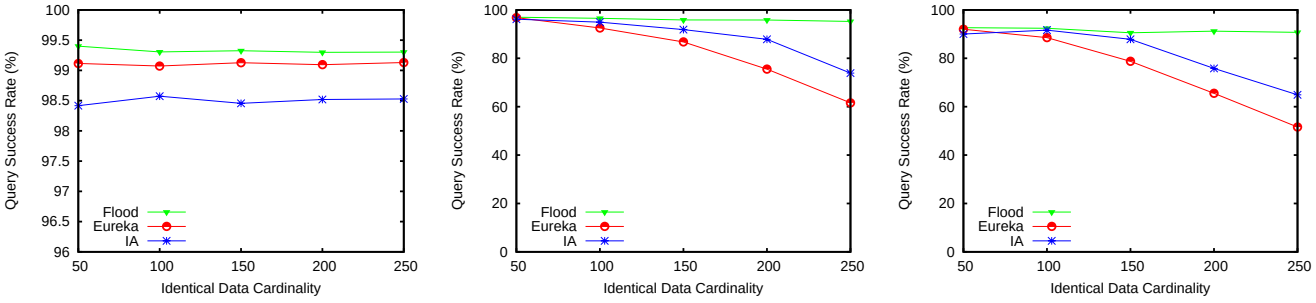
4.2 Query Traffic

We study the network traffic incurred by query propagation. We set the query rate $\gamma = 0.2$ and cache drop rate $\lambda = 0.035$. We randomly set the issuing time of the first query for each node in the range of $[0, 60s]$. We vary the data cardinality D from 50 to 250 and the network size N from 100 to 1000. The value of D affects the speed of IA value dissemination because a large D value provides more choices when a node issues a query, which introduces fewer cached data replicas of certain resources in the network. It sequentially slows down the dissemination of IA values, and indirectly lengthens the forwarding paths of queries. Therefore, in Figure 5, the query traffic of our IA approach increases. However, our approach outperforms the other two approaches in all cases. The flooding approach suffers more than one order of magnitude query traffic compared with



(a) 100 Nodes (b) 500 Nodes (c) 1000 Nodes

Figure 7: Performance of Average Query Response Time



(a) 100 Nodes (b) 500 Nodes (c) 1000 Nodes

Figure 8: Performance of Average Query Success Rate

the IA approach because IA only chooses the most promising neighbor to forward the query instead of flooding the query. Eureka also incurs much higher query traffic than the IA approach, because it blindly floods the query to its neighbors. Those neighbors then evaluate whether their information densities are larger than the former ones, and one or some of these neighbors take the forwarding process and flood the query to all its neighbors continuously, until the result is found. Although Eureka is much better than pure flooding, its blind flooding to the neighbors results in higher query traffic.

4.3 Communication Traffic

Figure 6 shows the results of communication cost which includes query traffic, result data traffic, index dissemination traffic and traffic incurred by incentive mechanism. The proposed IA approach significantly outperforms the other two competitors. Both Eureka and IA approach use indexes to maintain some useful information. However, Eureka incurs much more communication traffic than IA does, because of its inefficient index dissemination. It delivers updated index values to all neighbors whenever an update happens. When a neighbor receives this message, it updates its corresponding index value and recursively delivers the new updated value to its neighbors. Thus, every update in Eureka may involve all the nodes in the network. This is the main reason why Eureka incurs as much as or even higher communication traffic than the pure flooding approach does. In the IA approach, the indexes can be renewed during query process. A node sends the updated indexes to its neighbors in a fixed time interval, which does not incur recursive effort.

4.4 Query Response Time

In this set of experiments, we evaluate the query response time for successful queries. Adopting the typical policy in many mobile networks, we set a time threshold for queries, and assume a query fails if its issuer does not get any reply within the threshold. In Figure 7, we show the average query response time of all successful queries in the network by varying the network size and data cardinality. The flooding approach has the shortest response time owing to its fully distributed nature. Our IA approach is faster than Eureka, which means IA has shorter query paths than Eureka on average. As we can see, Eureka is a constrained distributed approach which propagates queries in a selective and parallel way. IA has only one but probably most optimal query path, thus it outperforms Eureka. This is owing to the advantages of its Intelligence Accuracy evaluation technique as well as the cooperation of each node. Meanwhile, the gaps among IA, Eureka and flooding are getting smaller when network size becomes larger, because the query paths of all three approaches get closer to TTL for larger networks.

4.5 Query Success Rate

Figure 8 shows the average query success rate for different methods. Clearly, the flooding approach performs best, as it always floods the query to all the neighbors, thus it will not miss any results stored in these neighbors. Compared with IA and Eureka, we find that IA is a little worse than Eureka in 100-node network, while it outperforms Eureka when the network size becomes larger. Although the proposed IA approach selects one path to forward the query, the IA value can effectively capture the information of data resources in

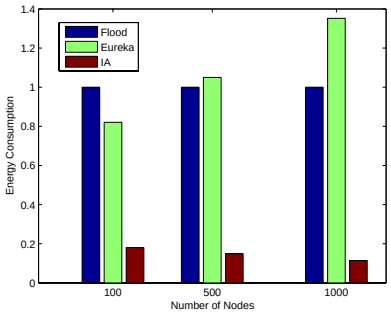


Figure 9: Average Energy Consumption

the network, and forward the query to the most promising neighbor. In such a highly dynamic environment, data and peers are frequently updated. The higher query success rate indicates higher scalability of our approach over Eureka.

4.6 Energy Consumption

Figure 9 shows the average battery energy consumption of each node in the network. We use the average energy consumption of the flooding approach as the baseline. The IA approach is much more efficient in energy saving than the other two approaches. Energy consumption is highly related to the total number of communication messages shown in Figure 6. More message transfer in the network consumes more energies. Eureka incurs more energy consumption than the flooding approach, because of its index dissemination and query propagation. In every single query step of Eureka, it floods the query to all neighbors. Although only one or a few neighbors with higher index values will further flood the query, many neighbors just do nothing but drop the messages, and hence the energies are wasted on these nodes. Additionally, the index update in Eureka involves the majority of nodes in the network, which introduces high energy cost.

5. CONCLUSIONS

In this paper, we have addressed the keyword based information retrieval problem in a popular and realistic file-sharing mobile peer-to-peer network. The proposed solution takes the advantage of the powerful mobile facilities to share resources and information which are of great benefit to people's daily life and various business applications. An intelligence-based information retrieval algorithm is proposed to help nodes decide where to forward the query, by mimicking different human behaviors in social networking. It is a light-weight and energy efficient approach using an Intelligence Accuracy (IA) mechanism to evaluate the distance from the node to the resource. We conduct extensive experiments to evaluate the performance of our proposed approach. The results show that our approach is well adaptive in highly dynamic mobile P2P network, and yields superior efficiency and effectiveness.

6. ACKNOWLEDGMENTS

This research was supported by the National Natural Science foundation of China under Grant No.60603045, the National High-tech R&D Program (863) of China under Grant No. 2007AA01Z153.

7. REFERENCES

[1] *The bluetooth specification.* http://www.bluetooth.com/dev/specifications.asp.

[2] *The IEEE 802.11 standards.* http://grouper.ieee.org/groups/802/11.

[3] *The Network Simulator-ns2.* http://www.isi.edu/nsnam/ns/.

[4] D. J. Abadi, S. Madden, and W. Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. In *VLDB*, pages 769–780, 2005.

[5] J.-Y. L. Boudec and M. Vojnovic. Perfect simulation and stationarity of a class of mobility models. In *INFOCOM*, pages 2743–2754, 2005.

[6] Z. Chen, H. T. Shen, Q. Xu, and X. Zhou. Instant advertising in mobile peer-to-peer networks. In *ICDE*, 2009.

[7] M. Conti, E. Gregori, and G. Turi. A cross-layer optimization of gnutella for mobile ad hoc networks. In *MobiHoc*, pages 343–354, 2005.

[8] S. Das, A. Nandan, and G. Pau. Spawn: a swarming protocol for vehicular ad-hoc wireless networks. In *ACM VANET*, pages 93–94, 2004.

[9] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale xml dissemination service. In *VLDB*, pages 612–623, 2004.

[10] M. Fiore, C. Casetti, and C.-F. Chiasserini. Efficient retrieval of user contents in manets. In *INFOCOM*, pages 10–18, 2007.

[11] A. Klemm, C. Lindemann, and O. Waldhorst. A special-purpose peer-to-peer file sharing system for mobile ad hoc networks. In *IEEE VTC*, pages 2758–2763, 2003.

[12] L. Liu, N. Antonopoulos, and S. Mackin. Social peer-to-peer for resource discovery. In *EICPDNP*, 2007.

[13] Y. Luo, O. Wolfson, and B. Xu. A spatio-temporal approach to selective data dissemination in mobile peer-to-peer networks. In *ICWMC*, page 50b, 2007.

[14] M. Lupur, J. Li, B. Ooi, and S. Shi. Clustering wavelets to speed-up data dissemination in structured p2p manets. In *ICDE*, pages 386–395, 2007.

[15] M. Motani, V. Srinivasan, and P. S. Nuggehalli. Peoplenet: engineering a wireless virtual social network. In *MobiCom*, pages 243–257, 2005.

[16] N.Chang and M.Liu. Optimal controlled flooding search in a large wireless network. In *WIOPT*, pages 229–237, 2005.

[17] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *MobiHoc*, pages 117–127, 2001.

[18] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. v. Steen, and H. Sips. Tribler: A social-based peer-to-peer system. In *Concurrency and Computation: Practice and Experience*, volume 20, pages 127 – 138, 2007.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. *ACM SIGCOMM*, 31(4):161–172, 2001.

[20] K. G. Siddhartha, M. Singh, D. Xu, and B. Li. Efficient peer-to-peer data dissemination in mobile ad-hoc networks. In *ICPPW*, page 152, 2002.

[21] J. Tchakarov and N.H.Vaidya. Efficient content location in wireless ad hoc networks. In *MDM*, pages 74–85, 2004.

[22] O. Wolfson, B. Xu, and R. M. Tanner. Mobile peer-to-peer data dissemination with resource constraints. In *MDM*, pages 1–15, 2007.

[23] B. Xu and O. Wolfson. Data management in mobile peer-to-peer networks. In *DBISP2P*, pages 1–15, 2005.

[24] X. Yang, H. B. Lim, T. M. Özsu, and K. L. Tan. In-network execution of monitoring queries in sensor networks. In *SIGMOD*, pages 521–532, 2007.

[25] Z.Cheng and W.Heinzelman. Flooding strategy for target discovery in wireless networks. *Wireless Networks*, pages 607–618, 2005.