PBQ: A Priority-Based Query Processing Algorithm in Opportunistic Wireless Sensor Network

Yongxuan Lai* and Guilin Li
School of Software, Xiamen University, Xiamen 361005, P.R. China

Abstract

Opportunistic wireless sensor network is a kind of opportunistic network that consists of nodes with sensing capabilities and actively or passively adopts the form of opportunistic transmissions. Due to the lack of stable end-to-end links and query-supporting infrastructures such as clustering or routing trees, the query processing in opportunistic wireless sensor networks is a challenging problem. In this paper, we propose a priority-based algorithm called PBQ for query processing in opportunistic wireless sensor network. Through the user-defined query priorities, the proposed algorithm properly selects the relay nodes and controls the forwarding of the messages. Meanwhile, the query priorities are adjusted dynamically so that the query results could be forwarded back to the query requester quickly and the residual query request messages could be cleaned up from the network, saving lots of unnecessary transmissions. Simulating results show that PBQ improves the overall utility of query processing. It could effectively improve the success rate of queries, and reduces the cost and delay of query processing in opportunistic wireless sensor networks.

Key Words: Query Priority, Query Processing, Opportunistic Network, Wireless Sensor Network

1. Introduction

Opportunistic wireless sensor network (OWSN) is a kind of opportunistic network that consists of nodes with sensing capabilities and actively or passively adopts the form of opportunistic transmissions. For example, in mobile wireless sensor networks, node movements and signal interruptions usually cause intermitted links and lead to opportunistic communications; and due to energy-reserving considerations, nodes may adaptively close their radio devices, leading a disconnected network [1]. End-to-end paths are not available in the opportunistic network and it adopts an opportunistic style of communication for message transmissions. Messages are temporarily stored on the nodes, and if there exists a chance of communication at some proper time, the messages are forwarded to other nodes and finally routed to their target nodes. There have been lots of applications in opportunistic wireless sensor networks, some representative applications include vehicle networking [2], wild animal tracking an monitoring [3], agricultural data gathering [4], message exchange in remote areas [5] and etc.

The sensor network is “data-centric” [6]. Nodes would continuously sense the environment and generate large number of sensing data, so the data processing and querying is a key issue in sensor networks. Classical query algorithms, e.g. TAG [6], PULL/PUSH [7], all depend on the network infrastructures such as routing tree and clustering to dispatch and collect the query-related messages. Yet as the opportunistic network lacks static links, similar network infrastructures supporting the query processing is usually very expensive and difficult to maintain. This brings about many challenging issues for the query processing in opportunistic wireless sensor networks, including the following aspects:

1. Every node could be the requester or responder of queries, and the query messages have no target addresses. The network could transmit the query mes-
messages only through the semi-flooding schemes, which usually lead to redundant and unnecessary message transmissions.

(2) The network exists multiple queries, yet only part of the queries could be processed properly due to the limited transmission chances and bandwidths. How to cooperatively answer the query and maximize the overall utility of network resource is a challenging problem.

(3) Opportunistic network adopts the “Store-Carry-Forward” schemes [8,9] for message communications. Due to the limited cache space, some data and query results may be discarded and lead to the loss of query results.

Queries have some semantic properties in various applications. For example, a query that detects whether a fire has occurred should return the query result as soon as possible. This kind of query is emergent and important. Yet a query that inquires the temperature of a node 1 hour ago is less emergent, it could suffer some delay or message loss. So according to the difference of the queries, if the query processing algorithm could allocate the network recourse and arrange the message forwarding properly, the query processing could be optimized and the overall result delay could be cut down.

In this paper we propose a priority-based algorithm called PBQ (Priority-Based Query processing) for query processing in opportunistic wireless sensor network. PBQ defines the query priorities based on user-defined properties, and properly selects the relay nodes and controls the forwarding and the amount of messages. Meanwhile, the query priorities are adjusted dynamically so that the query results could be forwarded back to the source node of the query quickly and the residual query request messages could be cleaned up from the network, saving lots of unnecessary transmissions and improving the overall utility of query processing. To the best of our knowledge, PBQ is the first step towards the research of priority based query processing in OWSNs. Simulating results show that PBQ could effectively improve the success rate of the queries, and reduces the cost and delay of query processing in opportunistic wireless sensor networks.

The rest of the paper is structured as follows: section 2 describes the related works; section 3 defines the network and query priority model; section 4 presents the detailed mechanism of the PBQ algorithm, including the setup phase, query forwarding phase, query executing and result returning phase, and query cleaning phase; section 5 describes the environmental setup and analyzes the simulation results; finally, section 6 concludes the paper.

2. Related Work

Opportunistic networks adopt a “Store-Carry-Forwarding” strategy for message transmission [8,9]. Messages are temporally stored on the nodes, and if there exists chance of communication at some proper time, the messages are forwarded to other nodes and finally routed to their target nodes. Epidemic routing [10] uses a flooding alike mechanism for message routing. Encountered nodes would fully take advantage of the communication chance for message exchange to increase the rate of successful message transmission and decreases the message delay. To cut down message transmissions and handle the network congestion, controlled-flooding algorithm [11] is proposed, which selectively forwards the message based on the forwarding probability and time-to-live (TTL) or kill time. It also adds a Passive Cure scheme to prevent the transmission of residual messages. In the prioritized epidemic routing PREP [12], messages are forwarded or deleted based on the current cost to destination, current cost from source, expiry time and generation time, and messages closer to the destinations would have more copies to improve the rate of successful transmission.

Besides the flooding alike scheme, another type of message forwarding scheme is to utilize the contextual information and knowledge of the network to optimize the message transmissions. ZebraNet project [3] uses a mechanism based on the history of node movements. Each node maintains a probability to the sink, and the node with higher probability would send its messages to the node with lower probability when two nodes encounter. Similar with [3], PROPHET [13] computes a forwarding probability based on the historical record of its observed contacts, and messages are routed to its neighbor only if it has higher probability than its neighbor. The CAR (Context-Aware Routing) scheme [14] takes the contextual factors such as residual energy, change rate of topology, and moving speed as the input, and uses
Kalman filter to calculate the probability to the destination node. Messages are sent to the node with the highest probability. Most of the schemes mentioned assume the messages of equal priority, and properties such as user-defined importance and time restraint for message transmissions are not defined for the optimization.

More recently, studies have focused on mobile social networks (special type of DTNs consisting of human-carried devices) and analyzed the social network properties of these networks to assist the design of routing algorithms, where data forwarding metric is centrality based. BUBBLE Rap [15] uses betweenness as the centrality metric which measures the social importance of a node facilitating the communication among other nodes. In friendship based routing [16], friendships in terms of their behaviors are defined between nodes (i.e. people) to facilitate message forwarding. And reference [17] studies the transient characteristics of node contact patterns. It formulates the transient social contact patterns of mobile nodes as a Gaussian function, based on which it develops data forwarding metrics to analytically predict the contact capability of mobile nodes with better accuracy.

Yet schemes mentioned above are general message transmission schemes, which are not optimized for the query processing in the opportunistic networks. There is not much work on the query processing in OWSNs. Reference [18] proposed a distributed information retrieval system that operates over a disruption tolerant network. Relevance scores and whether a document can be delivered successfully are estimated, and nodes retrieve top-ranked documents from their local collection and send them to the source of the query. DelQue [19] used semi-Markov process to compute neighbors’ social utility. It exploited the utility of each neighbor to represent its capability to query information and then collocate with the source to respond. Reference [20] presents a publish/subscribe system in opportunistic network. The core component of our system is to allocate the socially-aware brokers that are responsible for collecting subscriptions and dispatching messages. By dynamically controlling the number of brokers in the network, the system trades off between the efficiency and overhead. Different from [20], in this paper we assume messages are only related with the query priorities, and there are not brokers that are specially for subscription collection or dispatching. Every node could be the query requester and query replier.

3. Query Priority Model

We assume the network has N sensing nodes, every node has an unique id $s_i$. The network uses an opportunistic way for message transmission. When two nodes encounter, they establish a temporary communication link for message exchange.

The query requester generates a query request message $re$ and disseminates it to other nodes through opportunistic communications. The nodes who receive the request messages would make a data retrieval on its local storage. If the node could answer the query, it would forward the query result $rs$ to the query requester. Nodes who could answer the query request $re$ is called the query replier. There exists a mapping from the query and its results: $h(re) \rightarrow RS$. Here we assume every replier has all the data answering the query, which is denoted as $rs = RS$.

According to the restraint time and importance, queries could roughly be classified as the following categories:

1. High emergency, high importance.
2. High emergency, low importance.
3. Low emergency, high importance.
4. Low emergency, low importance.

Different type of queries require different amount of resources. Here we first define a standard query $S$.

If query $S(re, rs, st, sw)$ with default importance $sw$ could be successfully executed and the query result $rs$ could returned to the requester within the default time delay $st$, then $S$ is called a standard query.

The position of standard query $S$ is also illustrated in Figure 1. Based on $S(re, rs, st, sw)$, the priority of query

![Figure 1. Illustration of query processing.](image)
The priority of standard query 
never related with the query id’s frequencies of the encountered nodes. The priority of standard query 
P is positively related with w, yet adversely related with t. The priority of standard query 
(P(S(re, rs, st, sw)) is 1.0; for other queries, the priorities meet the following formula:

\[ 0 \leq P(Q(re, rs, t, w)) \leq \alpha^*T_1 + (1 - \alpha)^*T_2 \]

4. Priority Based Query Processing

4.1 Algorithm Overview

Query processing in opportunistic network would incur some time delay or even failure of processing due to the opportunistic style of communication. The key idea of PBQ is to cut off the time delay and improve the success rate of query processing through proper selection of relay nodes and controlling the amount of forwarding messages based on the query priorities. Meanwhile, query priorities are adjusted dynamically so that the query results could be forwarded back to the source node of the query quickly and the residual query request messages could be cleaned up from the network, saving lots of unnecessary transmissions and improving the overall utility of query processing. In general, the query processing contains 8 steps (as illustrated in Figure 2).

1. Query requester A receives a query from the user. It defines the query priority, distributes the priority according to the activeness of encountered nodes, and forwards the query request messages to the relay nodes.
2. The relay node receives the query, and forwards the message according to the message’s priority.
3. The relay node forwards the message to node B, who could answer the query.
4. Node B executes the query by searching its local storage and gets the query result.
5. Node B defines the priority of the query result, and forwards the result to the relay nodes.
6. The relay node receives the result message, and distributes the priority according to the activeness of encountered nodes, and forwards the query result to other relay nodes.
7. The relay node forwards the query result to the query requester A, the query is successfully processed.
8. Node A sends a “query-clean” command to clean the residual messages related to that query. The processing for the query ends.

In the following subsections, we categorize the 8 steps into 4 phases and describe the detailed algorithm of each phase. The 4 phases are the initial run, the query forwarding, the query execution and returning, and the query cleaning phase.

4.2 Initial Run

At the beginning PBQ adopts the epidemic alike schemes [10] for message transmission and the collection of network metadata. Every node maintains a contact list CL, which records the id’s and their frequencies of the encountered nodes. All the generated or received query messages are stored at the caches of the nodes. When there exists a communication chance, a node would filter and copy some messages to the message queue before forwarding them to other nodes. Cache is structured as a set of messages in the form of <query id, query, type, content>, where query is the detailed query description, content is the detailed data of the cache item, and type is the type of the cached message. Table 1 lists the type of messages.

![Figure 2. Message caching and forwarding model in node s.](image-url)
Meanwhile, based on the contact frequency and spare storage space, PBQ calculates an activeness factor $A$ for each node:

$$A(s_i) = \gamma \frac{N(s_i)}{N_{\text{max}}} + (1 - \gamma)\left(\frac{\text{usize}(s_i)}{\text{size}(s_i)} - 0.5\right) \quad (3)$$

where $\gamma$ is predefined balance factor, $N(s_i)$ is the number of nodes that could communicate with $s_i$ in the observed time span; $N_{\text{max}} = \max(N(s_i))$, $s_i$ is any node within the network. $\text{size}(s_i)$ is the total cache size of node $s_i$, $\text{usize}(s_i)$ is the unused cache size of node $s_i$. The activeness factor considers both the contact opportunities and spared cached storage. So nodes with large contact opportunities could shift some message relaying tasks to other nodes, and the overall communication and storage resources of the network could be properly utilized.

### 4.3 Query Forwarding

Every query request could be wrapped into one message. The message, e.g. $p$, contains the basic description about the query $Q(re, rs, t, w)$, and segments such as $<\text{Receiver}, \text{QP}, \text{Prior}>$. The Receiver segment is the prioritized receivers of the query result, so $\text{Receiver}$ is the node id of the query requester and its contact list ($\text{src} \cup \text{src . CL}$), $\text{QP}$ is the original priority when the query is issued, and $\text{Prior}$ is the priority of message $p$. $\text{Prior}$ is initially set to be the priority of the query: $\text{Prior} = P(Q(re, rs, t, w))$, yet it is dynamically updated when the message is disseminated.

When node $s_i$ encounters $s_j$ and gets a chance for communication, $s_i$ would exchange the query related metadata with $s_j$ and hence filters the messages to be transmitted. The metadata includes the query id and node activeness. As every query has a unique id within the network, node $s_i$ would filter the query messages that does not appear in node $s_i$ by comparing the messages stored in the caches of the two nodes. These messages are copied to the forwarding queue of $s_i$, as illustrated in Figure 2. Suppose $p1$ in node $s_i$ is the query request message, then $p1$ is forwarded in the following 4 steps:

1. $s_i$ copies $p1$ to $p2$ and sets the priority of $p2$. If currently the query semantic could decide $s_j$ could answer the query in $p2$, then $p2$ is set to have the maximal priority: $p2.\text{Prior} = \text{MAX}$, where $\text{MAX}$ is the maximal priority value a message may have. If the algorithm could not decide whether $s_j$ could answer the query, the priority of $p2$ is set as: $p2.\text{Prior} = p.\text{Prior} \times \frac{A(s_i)}{A(s_i) + A(s_j)}$. When $p2.\text{Prior}$ is greater than some predefined threshold: $p2.\text{Prior} > p1.\text{QP} * mp$, $p2$ is added to the forwarding queue, where $mp$ is a user-defined parameter.

2. Messages are sorted in the forwarding queue in $s_i$ based on their priorities, and $p2$ is transmitted from $s_i$ to $s_j$ according to its ranking. If there is not enough storage space in node $s_j$, PBQ would eliminate messages of the least priority, so the message of higher priority could be saved.

3. If $p2$ is transmitted successfully, message $p1$ in the cache of $s_i$ is updated accordingly: $p1.\text{Prior} = p.\text{Prior} \times \frac{A(s_i)}{A(s_i) + A(s_j)}$.

4. When the communication chance between the nodes ends, $s_i$ would dump its forwarding queue and all unsent messages are cleaned out.

Processing in node $s_j$ is the same as in node $s_i$. The priority of query request messages are distributed among the nodes according to the node activeness. Query messages of high priorities usually locate near the head of the forwarding queue, and they are forwarded earlier. Query messages of low priorities usually locate near the tail of the forwarding queue, or they are not copied to the queue. These messages they may not be able to be forwarded to other nodes. So according to the priority-based forwarding scheme, PBQ could automatically filter out the messages of low priority; through the priority threshold parameter $mp$. PBQ decides whether a message should be forwarded, and it finally controls the overall amount of forwarding messages.

### 4.4 Query Execution and Returning

When a node receives a query request message, it would search its local data to verify whether it could an-
answer the query. According to the query model defined in section 3, a node either has all the data or has no data to answer the query. If the node has no data answering the query, it would relay the request message to other nodes when there is communication chances; if the node has data answering the query, then it generates a query result message \( p' \) that warps the query result, and disseminates \( p' \) to the request node of the query.

The query result message \( p' \) is similar to the query request message, which includes the query metadata \( Q(re, rs, t, w) \) and \(<Receiver, QP, Prior, rs> \) segments, where \( rs \) is the query result. Message \( p' \) is added to the cache, and its initial priority is set: \( p'.Prior = P(Q(re, rs, t, w)) \). Different from the query request message, the result message \( p' \) has to be returned to the source node of the query, where the Receiver segment could guide the forwarding of the message. When node \( s_i \) encounters \( s_j, s_i \) would select messages, e.g. \( p3 \), that meet the following conditions from its cache:

\[
p3. Type = RS; s_j \in p3. Receiver
\]

where \( RS \) denotes the message type is query result. \( p3 \) is then copied as \( p4 \) and added to the forwarding queue. There are three cases for setting the priority and for the forwarding of messages:

\[
p4. Prior = \begin{cases} 
  MAX, & \text{if } sj = src \\
  p3. Prior \ast QP, & \text{if } sj \in rc.CL \\
  p3. Prior \ast \frac{A(s_j)}{A(s_j) + A(s_i)}, & \text{if } sj \notin p3. Receiver 
\end{cases}
\]

(5)

1. When node \( s_i \) encounters the query requester \( src (s_j = src) \), \( p4 \) is set to have the maximal priority \( MAX \) and send it to the query requester. The query is processed successfully and PBQ enters the query cleaning phase for that query (next subsection).

2. When node \( s_i \) encounters a node that belongs to the contact list of the query requester \( (s_j \in src.CL) \). The priority of \( p4 \) is temporally promoted \( QP \) times: \( p4. Prior = p3. Prior \ast QP \). Yet when \( p4 \) is successfully transmitted to node \( s_j \), the priority of the query result message is still distributed by the activeness of the nodes, that is: \( p3. Prior = p3. Prior \ast \frac{A(s_j)}{A(s_j) + A(s_i)} \) in node \( s_i \).

3. If \( s_i \) encounters other nodes \( (s_j \notin p3. Receiver) \), then \( p4 \) is forwarded based on the activeness of the encountered nodes. Meanwhile, \( p4 \) could be used to clean part of the residual messages for that query. The query request messages, e.g. \( p5 \), that meet the following conditions in the cache of \( s_i \) can be deleted:

\[
p5. Type = RE; QueryId(p5) = QueryId(p4)
\]

where \( RE \) denotes the message type is query request, \( QueryId(p) \) gets the query id of the message \( p \).

4.5 Query Cleaning

When the query result is routed to the query requester, the query is successfully executed; yet there are still some outdated messages left within the network, which include the request messages or result messages of the successfully executed query. So a query cleaning phase is needed to clean the residual messages and prevent redundant transmissions.

For the query requester \( s_i \), when it receives the query result message \( p4 \) from node \( s_j \), it would generate a query cleaning message \( p'' \). The type of message \( p'' \) is set to be \( KQ \), which indicates the message is a cleaning message. It includes the id of the query to be cleaned, and its priority is set to be the maximal: \( p''. Prior = MAX \). Message \( p'' \) is then immediately added to the node cache and routed back to \( s_j \) and forwarded to other nodes. The message priority is distributed similarly as the forwarding of query request messages described in subsection IV.C.

When a node receives the cleaning message, it would delete all messages related to that query from its cache and the forwarding queue. So when node \( s_i \) receives message \( p'' \), \( s_i \) would delete messages that meet the following conditions:

\[
QueryId(p) = QueryId(p'')
\]

(7)

As message \( p'' \) is initially set to be of very high priority, it could be quickly disseminated among the nodes, so the query cleaning process is accelerated to prevent transmissions of outdated messages. It is to be noted that due to the priority distribution mechanism, the priority of a message in PBQ would gradually decreased
when the message is forwarded to other nodes. As each node replaces its cache through the priority of the messages; so finally, some messages would be automatically deleted from the cache because their priorities are too low. In this way, messages of low priorities would not be disseminated to other nodes, and unnecessary transmissions are saved.

## 5. Experimental Analysis

### 5.1 Environment Setup

We implement the PBQ scheme in C# to verify its efficiency and effectiveness. Noting that nodes equipped by human or wild animals would stay and get together at a “community” for most of the time, we first define a group-based node mobility model: nodes are divided into $K$ groups, and each group has a predefined area called group area. Nodes walk inside the group area with a probability of $g_i$ according to the Way-Point mobility model, and outside the group area with probability of $1-g_i$. Node would sense the environment and generate a tuple of the temperature every 1.0 seconds, yet only the latest 100 tuples are recorded in its local storage. The query is defined as: given the node id, return the top 10 highest temperature readings. For the standard query, the restraint time is 200s and the priority is defined 1.0; for other queries, the restraint time $t$ and priority $w$ is greater than 0 and follows a normal distribution: $t \sim N(200,200)$, $w \sim N(1,0.5)$. Every node generates a query at an average rate of 0.03 query/s.

As the cost of communication dominates the depletion of the limited battery energy in sensor nodes, we present only the total communication cost (number of packets) incurred by various algorithms. And we assume ideal links when two nodes meet and establish a connection. The query request, query result and metadata could be wrapped into one message respectively. Table 2 lists the default parameters in the simulations.

We also compare PBQ with other algorithms. Yet because there are few algorithms specially designed for the query processing in OWSNs, the compared algorithms are based on the data forwarding algorithms such as Epidemic [10] and PROPHET [13], where queries are forwarded to the replier, and the query results are routed back to the query request node. They adopt a “first-in-first-out” scheme for the cache replacement. Simulating results show that compared with other algorithms, PBQ could effectively improve the success rate of queries, and reduces the cost and delay of query processing in opportunistic wireless sensor networks.

### 5.2 Experimental Analysis

The amount of queries is represented by the query rate, which is the average number of queries a node generates per second. As illustrated in Figure 3, message transmissions grow with the query rate for all algorithms. When more queries are generated, more messages are disseminated to process the queries. The cost of PBQ is the minimal among all the compared algorithms, which is about 72–81% of the cost in Epidemic. This is because in Epidemic nodes would take advantage of every communication chance to exchange messages. Yet PBQ adopts a priority-based scheme, where messages with low priority would not be exchanged when nodes contact each other. And there is a query cleaning phase to clean the outdated messages, saving lots of unnecessary messages.

Figure 4 depicts the impact of query rate to the success rate of queries. The success rate decreases as the query rate grows up for all the algorithms. The success rate of PBQ is about 63%, which is more than 5% higher than Epidemic and PROPHET schemes. The difference is even higher (up to 18%) when there are more queries within the network. This is largely because the priority

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>50</td>
<td>Number of nodes in the network</td>
</tr>
<tr>
<td>$W$</td>
<td>90 m</td>
<td>Simulation Field: 90 m*90 m</td>
</tr>
<tr>
<td>$K$</td>
<td>10</td>
<td>Number of groups</td>
</tr>
<tr>
<td>$R$</td>
<td>6 m</td>
<td>Radio range of the nodes</td>
</tr>
<tr>
<td>Sim_T</td>
<td>2000 s</td>
<td>Total simulation time</td>
</tr>
<tr>
<td>Min, max</td>
<td>1, 5 m/s</td>
<td>Min/max speed of the nodes</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>[5,50]</td>
<td>Bandwidth of the nodes (packets/s)</td>
</tr>
<tr>
<td>$g_i$</td>
<td>0.6</td>
<td>Probability of nodes walking in the group area</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.5</td>
<td>Index when calculating the priority</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.7</td>
<td>Index when calculating the activeness factor</td>
</tr>
<tr>
<td>$mp$</td>
<td>0.1</td>
<td>Threshold of priority when forwarding a message</td>
</tr>
<tr>
<td>MAX</td>
<td>10</td>
<td>Maximal value of priority</td>
</tr>
<tr>
<td>BufferSize</td>
<td>200</td>
<td>Size of cache (packets)</td>
</tr>
<tr>
<td>QueueSize</td>
<td>30</td>
<td>Size of forwarding queue (packets)</td>
</tr>
</tbody>
</table>
based message forwarding and caching in PBQ facilitates the speedy diffusion of query request messages and query result messages. The compared schemes treat all messages to be evenly important, and lots of query messages would not be transmitted or even be dropped because of the lack of cache space.

The time delay of a query is defined as the time span that begins at the time when the query is issued and ends at the time when the query result is successfully returned to the requester. In Figure 5 line A-a denotes the overall time delay, which is the average time delay of all issued queries including the successful and failed queries; line A denotes the average time delay of successful queries. From the figure we could see the overall time delay increases with the query rate. This is largely because there are more failed queries when there are more queries in the network. If the query is failed, the delay is set to be the length of the simulation, which is 2000s. So as PBQ has the highest success rate, its overall time delay is the smallest. For the successful queries, the average time delay of the algorithms all lie in the range of [220s, 300s], which are close to each other.

As the query processing schemes in opportunistic network are based on a “Store-Carry-Forward” mechanism for the message forwarding, in the simulation we also analyze the impact of cache sizes. From Figure 6 we could see that the number of message transmissions increase with the cache size for all the algorithms. Epidemic takes advantage of every communication opportunity for message transmission. If the cache is larger, more messages are stored in local storage, and more messages are exchanged between the encountered nodes. So in Epidemic, the number of transmissions when the cache size is 500 is about 80 percent more than that when the cache size is 100. Yet PRPHET and PBQ set thresholds for the message communication: when the difference of probability in PROPHET is below the threshold, there would not be message exchange among the nodes.
when the message priority is below the threshold, the message would not be send among the nodes. So for those two algorithms, the increase of message transmissions is relatively less as the cache size grows. For PBQ, the number of transmissions when the cache size is 500 is about 25 percent more than that when the cache size is 100.

Figure 7 depicts the impact of cache size to the success rate of queries. When nodes have larger cache storage, more query request and result messages could be stored in the nodes, and the possibility of routing these messages to the target node also becomes larger. So from the figure we could see the success rate increase with the cache size for all algorithms. The success rate of PBQ is more than 8 percent higher that of the compared algorithms. This is largely because of the priority based message forwarding mechanism. Messages with high priority are quickly disseminated to other nodes to increase the probability of reaching the target nodes who could answer the queries. Meanwhile, PBQ takes the spare cache space into consideration to calculate the node activeness. Nodes with large encountered nodes may shed some of the message storing and forwarding to other nodes. Yet for Epidemic and PROPHET, the node with larger encountered nodes would receive and store large number messages from other nodes. The cache space is quickly used up to trigger a cache replacement process, and some messages have to be deleted before being disseminated to other nodes. So some request messages or result messages would be lost, which harms the success of the query processing.

As described in equation 3, the activeness index $\gamma$ is set as a balance factor between frequency of encounters and spared storage space. The balanced index $\gamma$ is predefined by the users, yet we also constructed experiments to study its optimal value. From Figure 8, we could see that the transmissions decrease when the index $\gamma$ goes down. When $\gamma$ is 1.0, the activeness is only determined by the frequency of encounters, and messages are more likely to be exchanged among the active nodes. So the algorithm has the largest number of transmissions, and some nodes would exhaust their buffer and shed some of the messages. When $\gamma$ becomes smaller (e.g. 0.6–0.9), the spared storage space has more weight on setting the activeness, and messages are more likely to be shifted to nodes with larger spare storage space. This makes the query success rate become larger as some otherwise lost messages could be stored to nodes inside the network. The query success rate goes to the maximal (0.63) when $\gamma$ is around the range of 0.6–0.7. When $\gamma$ is set too small (e.g. < 0.4), the spare space have greater weight on setting the activeness. Then message are more likely to be gathered to the nodes with large buffer space because their activeness is high, yet those messages may not be forwarded to other nodes due to fewer encounters of the nodes. So here we set $\gamma = 0.7$, where PBQ get a relatively good performance for all the experiments.

Finally, we also analyze the percentage of query request messages in the transmissions. As illustrated in Figure 9, the request percentage is higher than 68% and grows with the query rate for all the algorithms. When more queries are generated, more query requests are disseminated among the nodes. Part of the request messages would be dropped because of the cache replacement mechanism or because they do not encounter the query repliers. So there are more query request messages than the
query result messages, and cutting down the number of request messages is a key issue for the query processing algorithms. The query cleaning phase in PBQ deletes the outdated request messages, which partly cuts down the overall message transmissions. Also, the request percentage in PBQ is about 6% higher than the compared algorithms. This is because the query result message contains a contact list of the query requester. The query result messages hence have target nodes and are disseminated according to their CL’s. This reduces the number of query result messages and reduces the percentage of request messages.

6. Conclusions

As the integration of opportunistic networks and wireless sensor networks, query processing is becoming an important issue in OWSNs. In this paper we have proposed a priority-based algorithm called PBQ for query processing in opportunistic wireless sensor network. The algorithm contains the initial run, the query forwarding, the query execution and returning, and the query cleaning phases. The key idea of the algorithm is to define the query priorities based on user-defined properties, and to properly select the relay nodes and control the forwarding and the amount of messages. Meanwhile, the query priorities are adjusted dynamically so that the query results could be forwarded back to the source node of the query quickly and the outdated query messages could be cleaned up from the network, saving lots of unnecessary transmissions and improving the overall utility of query processing. Simulating results show that PBQ could effectively improve the success rate of queries, and reduces the cost and delay of query processing in opportunistic wireless sensor networks.

For the future work, we are to investigate the impact of cache strategies and data redundancy to the query processing algorithms, and to optimize the algorithms in the OWSNs. Meanwhile, we are to extend the type of queries in the network, and propose the query algorithms accordingly to improve the overall usage of the network.

Acknowledgements

This work is supported by the Natural Science Foundation of China (No. 61202012, 61100032); Fundamental Research Funds for the Central Universities (No. 2012121030).

References


Manuscript Received: Jan. 1, 2013
Accepted: May 23, 2014