International Journal of Innovative
Computing, Information and Control
Volume 8, Number 5(A), May 2012

ICIC International ©2012 ISSN 1349-4198
pp. 3081–3102

BIT COLLISION DETECTION BASED QUERY TREE PROTOCOL FOR ANTI-COLLISION IN RFID SYSTEM

Haosong Gou and Younghwan Yoo

School of Computer Science and Engineering
Pusan National University
Busandaehak-ro 63beon-gil, Geurnjeong-gu, Busan 609-735, South Korea
{ gouhaosong; ymomo }@pusan.ac.kr

Received February 2011; revised August 2011

Abstract. *In radio frequency identification (RFID) systems, the anti-collision protocol is a key topic that has attracted a great research interest. Those protocols can be divided into two categories: ALOHA-based and binary query tree (QT) based algorithms. The ALOHA-based protocols avoid collisions by distributing tags into different stochastic timeslots. In contrast, the QT-based protocols utilize prefix matching techniques, in the meanwhile, achieving the reliable identification throughput. This paper proposes a kind of QT protocol, called enhanced BQT (EBQT), which is based on our previous BQT (Bit collision detection based Query Tree) [19]. In addition to the employments of individual bit collision detection mechanism, the new protocol also uses the correlated information between two sequential identification to accelerate its process. Analysis and simulation show that EBQT reduces tag collisions, which accelerates the tag identification process for mobile tag identification.*

Keywords: AIS, Binary tree protocol, BQT, EBQT, MQT, RFID system

1. **Introduction.** Recently, the application of radio frequency identification (RFID) has become very popular in many areas, such as service industries, procurement and distribution logistics, and manufacturing industry. The RFID technology provides the automatic identification of people, animal, goods, and products in transit [1]. With the success of this technology, intensive researches [2-5] have been aspired to resolve the issues from the actual application.

An RFID system typically consists of three basic components: tags, a reader, and a data processing server. The reader collects data at a certain radio frequency (RF) and identifies all tags within its range. For tag identification, one-to-one communication between tag and reader can completely avoid collisions in theory. However, this obviously increases time needed for identification. Therefore, the reader capacity to process many tags at a time is crucial for fast data collection. The tag processing efficiency can be improved as more and more tags are identified simultaneously. However, concurrent messages from multiple tags would lead to collisions, resulting in the failure of tag identification. Furthermore, retransmission message from those tags may experience collisions again, leading to the remarkable increase of tag identification latency and RFID system performance degradation.

In order to address this challenge, various anti-collision algorithms [2-21] have been proposed recently. While developing a practical RFID system, we devised an efficient tag identification protocol for the retail warehouse. In our target logistics environment, we are supposed to track all items by a mobile reader as shown in Figure 1. Moving straightly, the reader identifies tags within its communication range. The practical application requests

our system to meet two main requirements: 1) small size and low-cost tags; and 2) a fast and reliable identification algorithm to recognize tags.

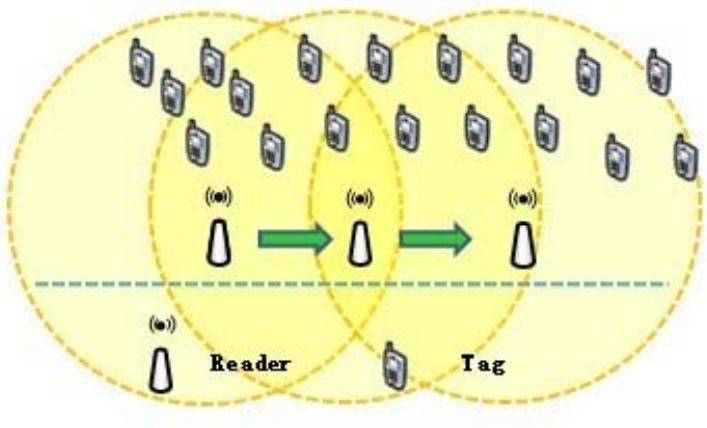

FIGURE 1. Mobile reader for tag identification

RFID tags fall into two categories: active and passive (memoryless). An active tag is equipped with a battery and relatively expensive circuit. Thus it can perform complicated processing, but its size is large. On the other hand, a passive tag is cheap, small, and powered by the radio from reader, but it has no storage for processing history, except for the ID and some embedded basic information. In usual, active tags are used for large goods such as containers in a port or cars in a parking area, while passive tags for small items such as goods in a retail warehouse.

Besides the hardware features of an RFID system, a tag identification algorithm is also critical. Existing algorithms can be divided into two categories: ALOHA-based probabilistic algorithms and binary query tree (QT) based algorithms. The main idea of both the algorithms is to split numerous tags into smaller subsets with different transmission times to reduce or avoid collisions. In the ALOHA-based algorithms, each tag experiencing collision is delayed for retransmission with a stochastic timeslot, which reduces a collision probability during the retransmission [6-8]. However, the high throughput cannot be guaranteed in such schemes. This throughput problem has been addressed by many binary tree search algorithms [9-17]. These algorithms split collided tags into smaller subsets by extending tag ID prefixes until only one tag remains in each round of the tag-reader communication.

The goal of this paper is to develop a binary QT protocol for memoryless tags for logistics in retail warehouses. Considering the communication overhead, this paper proposes a new tag identification protocol called EBQT (enhanced BQT). This is based on our previous work, BQT (Bit collision detection based Query Tree), which adopts a hybrid Manchester coding to accelerate identification process.

This paper has three main contributions:

(1) Although many query tree based protocols have been proposed to improve the performance of an RFID system, only MQT considers mobile tag identification. Our research could enrich the improvement in this field.
(2) Compared with the previous query tree based protocols, EBQT has shorter identification time delay and lower communication overhead in both static and mobile identification scenarios. Also it decreases the cost in other aspects as well, such as computing complexity and spatial complexity.

(3) Compared with the previous works employing the Manchester coding, EBQT resolves the existing problems addressed in Section 2 to make itself applicable to our practical application.

The rest of this paper is organized as follows: Section 2 reviews related works. Sections 3 and 4 describe the implementation of our previous BQT and new proposed EBQT protocol. Section 5 gives the mathematical performance analyses. Section 6 compares the performance of the proposed algorithm with previous improvements of the QT protocols. Sections 7 and 8 conclude this paper and address some future works.

2. **Related Work.** Binary query tree (QT) protocol [11] is considered as a milestone in the development of binary tree-based algorithms for passive tags. Although the QT protocol can guarantee reliable performance, it needs a long time to complete the identification process. In order to shorten the processing time, several enhanced QT protocols have been proposed: adaptive memoryless QT (MQT) [13], anticipative inquiry scheme (AIS) [14], prefix-randomized QT (RQT) [15], hybrid QT (HQT) [16] and QT-based reservation (QTR) protocol [17]. Among them, RQT, HQT, and QTR employ some features of active tags, and thus are not applicable to memoryless tags any more.

In addition to aforementioned protocols, other enhanced algorithms in [1,18-20] try to achieve higher performance by adopting the Manchester coding [1]. However, in [1,18], the authors assumed that the tags identified already would not give a response even if their IDs accord with the prefix in the current query. Because they used stored information on the previous processes, they can be implemented in active tags only. On the other hand, Liu et al. [19] aims at passive tags, but its scheme suffers from significant performance degradation. In this scheme, based on tag responses received during a communication round, a reader generates all possible IDs for consequential queries. For instance, if the string "1**" is detected from all tag responses, the reader generates the candidate IDs "100", "101", "110", and "111". It works well in case of a small number of bit collisions. However, the number of candidate IDs to be generated increases along with the number of collided bits. Thus if collisions happen at many successive bits, the performance may be aggravated seriously. Finally, the proposal in [20] employs extra bits to reconstruct tag ID and adopts the Manchester coding scheme to accelerate the identification process. However, ID structures of all our tags are pre-assigned by the vendor in accordance with the global standard, so it is impossible for us to reconstruct them. Meanwhile, a kind of masking code "x" is used in this proposal, but it does not suggest how to implement this code practically.

Based on the above description, only MQT and AIS are selected as benchmarks to be compared with our research.

2.1. **Adaptive memoryless QT (MQT) protocol.** Myung et al. [13] enhance the QT protocol with a new practical assumption; they aimed to accelerate identification by making use of the information from the previous process.

The MQT also conducts the same rounds of queries and responses as the QT protocol, except that in the MQT initial process, a reader establishes a queue Q for extended prefixes and a candidate queue CQ to shorten the execution time of the next process. All prefixes causing collisions are pushed into Q, but prefixes receiving no or one reply are pushed into CQ. Q maintains queries for the current identification process, while CQ compiles queries for the next process. During the first process, MQT works the same as the QT protocol does. From the next process, the reader updates Q by CQ, and dequeues a query from Q one by one before transmitting it. The tag identification process continues until Q is empty.

After the first non-initial process, abnormal prefixes have to be deleted in each process. This is to traverse the tree and find the least and shortest prefixes for re-identifying current set of tags.

2.2. **Anticipative inquiry scheme (AIS) protocol.** Hsu et al. [14] propose another type of QT protocol called Anticipative Inquiry Scheme (AIS). All tag IDs appear at leaf nodes of a binary ID tree, and the IDs are randomly and uniformly distributed between the left sub-tree and right sub-tree. As a result, the left and right sub-trees have the similar number of leaf nodes, and the information achieved during the process in the left sub-tree to accelerate the identification process in the right sub-tree.

According to the principle of AIS, it keeps tracking the identification process and records the number of collisions of each level in the left sub-tree. Then, prior to identifying the tags in the right sub-tree, a threshold on the number of collision is set to move the identification process to the deeper level directly, instead of scanning the right sub-tree level by level from the root node ('1'). In other words, when identifying the IDs of the right sub-tree, AIS sends longer prefixes, such as "100" and "110" to query the tags according to the threshold, not starting from a shorter prefix, "1". However, the author does not give the academic way to set the collision threshold optimally. AIS is hard to be applied in practical application.

3. **Proposed BQT Protocol.** As mentioned earlier, this paper proposes an enhanced protocol based on our previous BQT [21]. This section introduces the previous BQT before describing the enhanced version in the next section.

3.1. **Network model.** Assume that there are numerous passive (memoryless) tags to be identified.

(1) Each tag has a unique ID consisting of "0" and "1", and the ID length is K bits.

(2) From when a reader starts work until all tags are identified, is called one "process". The reader has to be stationary until one process is over.

(3) Each tag in a reader's range can communicate with the reader directly without communication errors.

(4) Receiving a query, all tags send their responses simultaneously. Because of the very short communication range (3-5 m in our study) and the ultra-high radio transmission speed, the difference between the arrival times at the reader from all tags to the reader can be ignored.

3.2. **Network model.** In our proposed algorithm, the bit collision detection is used to achieve fast identification process. As shown in Figure 2, two tags collide with each other at the first and fourth bits, but the second and third bits can be successfully received.

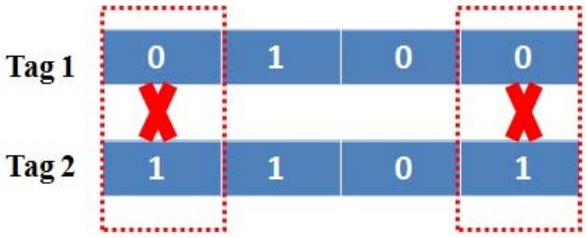

FIGURE 2. Bit collision detection

In order to achieve individual bit collision detection, we adopt the Manchester coding system [1]. If two (or more) transponders simultaneously transmit bits of opposite values,

then the positive and negative transitions cancel out each other. As a result, an invalid combined signal appears as shown in Figure 3, being recognized as a bit error. On the other hand, a valid combined signal is created if transponders transmit the same bit.

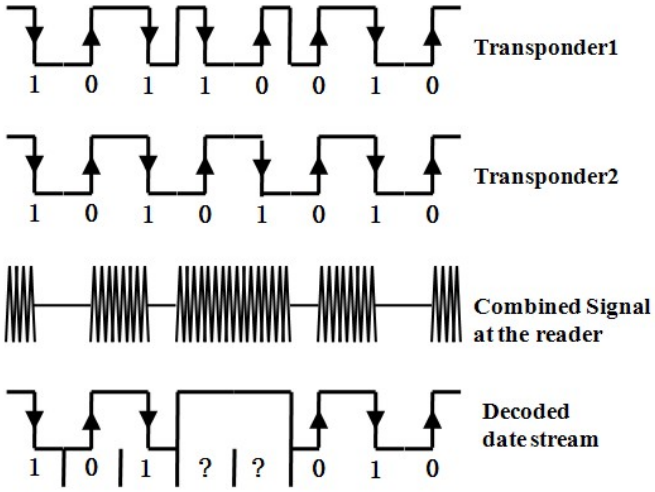

FIGURE 3. Integrated signal of Manchester code [1]

In the proposed algorithm, a reader has to send masking code "**...*" in the first round and also mark the collided bits by the masking code "*". Thus, we designed a hybrid coding to integrate the code "*" into Manchester coding system without extending the current coding length. In our hybrid coding, there are three different situations, as shown in Figure 4.

(1) The masking code "*" that does not appear at the first bit. The voltage stays at the same level as the end of the last bit code.

(2) The masking code "*" that appears at the first bit. The voltage level continues, which is low in our system.

(3) All bits are the masking code "*". The voltage stays at the low level.

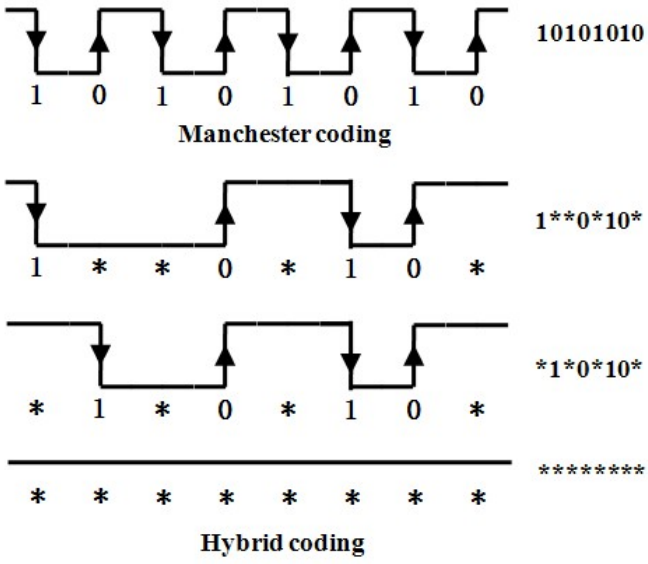

FIGURE 4. Hybrid Manchester code

3.3. **Protocol of BQT.** The previous BQT [21] works well in a situation where all tags are static, consisting of the following steps.

(1) The reader initializes a query string by $\underbrace{***...*}_{k}$ (note that each "*" is one-bit wild masking for the value "0" or "1") and an empty queue Q to track the identification process. Then it pushes an initialized query string into Q.

(2) At the beginning, the reader fetches a query string from Q and broadcasts it to all tags, and each tag sends back a response to the reader. All tags give responses simultaneously, and collisions occur.

(3) When collisions occur, the reader detects the individual bit collisions through our new hybrid coding scheme and updates the query string. There are two different cases of individual bit detections as shown in Figure 5.

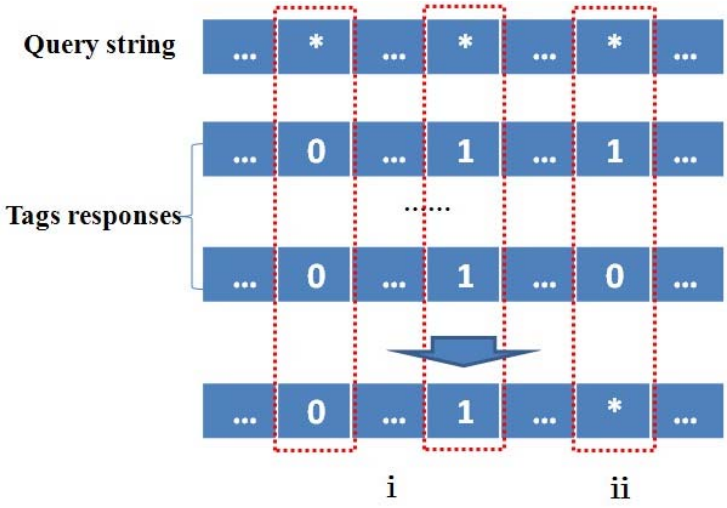

FIGURE 5. Individual bit collision detection

i. At a specific bit position, if the values in all responses are "0" or "1", no collision occurs at this bit and this bit of the query string can be respectively updated by the corresponding value, "0" or "1".
ii. At a specific bit position, if the values in some responses are "0" but others are "1", a collision occurs, and this bit of the query string remains "*".

(4) After the query string is updated, there exist the following three situations:

a. Multiple "" exist in the updated query string.* The reader replaces the first "*" with "0" and "1" respectively to create two new strings, then pushes them into the queue.

b. Single "" exists in the updated query string.* The reader replaces this "*" with "0" and "1" respectively, and marks them as two successfully identified tags without extra queries. Obviously, the one bit collision means there must be two tags with IDs differentiated at this bit.

c. No "" exists in the updated query string.* Once the updated query string does not include any masking code, it is an identified ID.

Example:

a. More than one "*" exist. An example of an updated query string is "1**0"; the first "*" is replaced with "0" and "1" respectively, resulting in two new query strings, "10*0" and "11*0". Then, they are pushed them into the queue.

b. Only one "*" exists. If the updated query string is "10*1", tags 1001 and 1011 are identified without an extra query string. It saves two times of query transmissions.

c. No "*" exists. If the updated query string is 1100, that means tag 1100 has been identified.

(5) Queue status is checked.

a. Queue is not empty. If the queue is not empty, continue to step (6).

b. Queue is empty. If the queue is empty, the identification process is terminated.

(6) The reader gets the next query string from the queue and sends it to all tags.

(7) In order to minimize communication overhead, in our solution, all tags matching the query string only respond with the bit values at the position corresponding to "*" in the query string.

Example:

In Figure 6, the reader sends query string "1*0*" to all tags. Then tags whose first bit is "1" and third bit is "0" send a response with their own values for the unknown two bits.

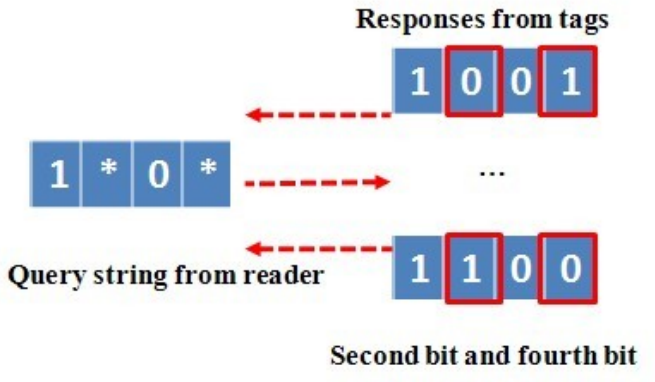

FIGURE 6. Corresponding bits response

(8) After receiving responses, the reader restores the received bits to their corresponding position or detects individual bit collision. After that the query string is updated. The multiple replies and single reply update case is illustrated in Figures 7 and 8.

a. Multiple replies. More than one tag give responses, leading to collisions. The reader detects bit collisions and updates the query string, as shown in Figure 7, then continues to step (4).

b. Single reply. Only one tag gives a response, the reader replaces '*' in the query string with the received bits and regards it as an identified tag, as shown in Figure 8. Then go to step (6).

4. Enhanced BQT Protocol.

As mentioned earlier, our system targets tag identification using a mobile reader in a retail warehouse. Thus between two consecutive identification processes, some tags may newly arrive in and some existing tags may leave out of a reader's range. The tag movement is considered in BQT also, but EBQT enhances it to get a better performance by using the information of one process for the quick completion of the next process.

4.1. Protocol of EBQT.

MQT [13] also takes advantage of the current information for the next process. In MQT, the prefixes getting no replies (***Idle prefix***) are reserved to identify newly arriving tags. Also, the prefixes getting one reply (***Identified prefix***) are separately reserved to identify the remaining tags. These subdivided prefixes speed up the identification process, because it does not have to start from the root of the ID binary

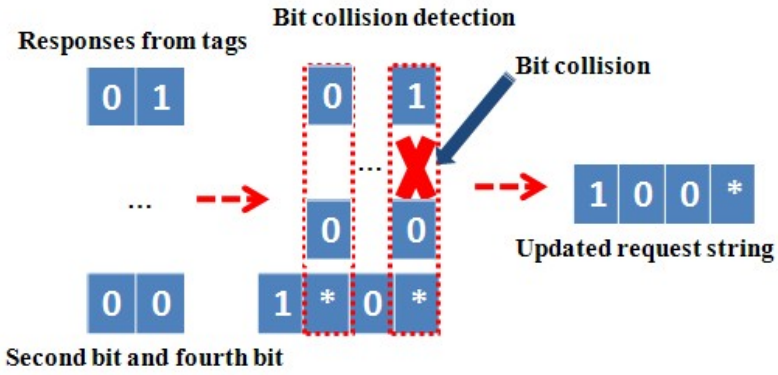

FIGURE 7. Multiple replies update

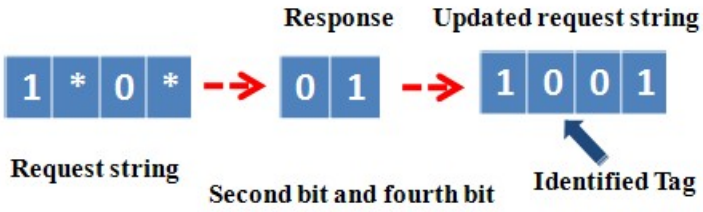

FIGURE 8. Single reply update

tree any more. However, the performance of MQT significantly depends on how much two consecutive processes are different from each other. Large difference may not shorten time delay for identification process at all.

Actually, the number of tags in the initial process itself can be used to shorten future processes in condition that the number of tags does not fluctuate in every process. If the rough number of tags is known to a reader, it can expand in advance the ID prefixes in query messages to an enough level to reduce collisions. For example, if 5 tags are expected, the prefix in the first query may start from the second level like $00*\ldots*$, $01*\ldots*$, $10*\ldots*$, and $11*\ldots*$ instead of the root level $***\ldots*$.

Based on this observation, this paper proposes an enhanced BQT protocol called EBQT to achieve the lower and consistent time delay for tags identification in mobile scenarios.

The difference of EBQT from the BQT protocol can be seen in detail in the following steps:

(1) Firstly, an ID counter is setup to count the number of identified tags. Whenever a tag ID is identified, the counter increases by 1.

(2) After the initial identification, a reader begins to generate prefixes for next processes. The prefixes can be generated by the following steps:

a. The starting level is computed according to the following equation:

$$L = \lfloor \log_2 IC \rfloor$$

where, IC is the number of identified tags in the previous process.

b. The reader generates the prefixes from $\underbrace{00\ldots0}_{L}*\ldots*$ to $\underbrace{11\ldots1}_{L}*\ldots*$ and pushes them into Q.

c. IC is reset to 0.

(3) BQT steps (2) $\sim$ (8) are performed for the next process.

4.2. **Working example of MQT and EBQT.** In order to understand and compare MQT and our EBQT, we walk through the initial and non initial process of both the algorithms in Tables 1-4.

4.2.1. *Initial identification process.* Assume that there are four tags to be identified and that their IDs are 0100, 0110, 1000 and 1001.

TABLE 1. Initial identification process of MQT

Rounds	prefix	Response	Result	Q	CQ
1	#	0100,0110, 1000,1001	Collision	0,1	
2	0	0100,0110	Collision	1,00,01	
3	1	1000,1001	Collision	00,01,10,11	
4	00	Idle		01,10,11	00
5	01	0100,0110	Collision	10,11,010,011	00
6	10	1000,1001	Collision	11,010,011,100,101	00
7	11	Idle		010,011,100,101	00,11
8	010	0100	Identified	011,100,101	00,11,010
9	011	0110	Identified	100,101	00,11,010,011
10	100	1000,1001	Collision	101,1000,1001	00,11,010,011
11	101	Idle		1000,1001	00,11,010,011,101
12	1000	1000	Identified	1001	00,11,010, 011,101,1000
13	1001	1001	Identified		00,11,010,011, 101,1000,1001

TABLE 2. Initial identification process of EBQT protocol

Rounds	Query string	Response	Updated string	Q	Identified tags
Initialization				****	
1	****	0100,0110 1000,1001	****	0***,1***	
2	0***	100,110	01**	1***,010*, 011*	
3	1***	000,001	100*	010*,011*	1000,1001
4	010*	0	0100	011*	0100
5	011*	1	0110		0110

Tables 1 and 2 demonstrate the difference of MQT and EBQT in the initial identification process. From both the tables, we can see that EBQT outperforms MQT. The reasons are: (1) Bit collision detection is adopted in EBQT, so it can avoid idle responses. As shown in the tables, MQT meets four idle responses (Rounds 4, 7, 8, 11 in Table 1), but there is no idle response in EBQT. (2) MQT expands the prefix bit by bit whenever a collision occurs, but EBQT can expand several bits at once to the collided bit position to avoid repeated collisions. For example, to identify tag 1000 and 1001, MQT expands the prefix three times (Rounds 3, 6, 10 in Table 1) while EBQT extends just once (Round 3 in Table 2). (3) Due to the bit collision detection, EBQT can identify two tags simultaneously if there is only one bit collision. For example, in Round 3 in Table 2, 1000 and 1001 make only one bit collision, so they can be identified directly without extra queries.

4.2.2. *Non-initial identification process.* After the initial process, we assume that tags 0100 and 1001 move out of the reader's communication range and that tags 1100 and 1101 move into the reader's range. The new tag ID set contains 0110, 1000, 1100 and 1101.

TABLE 3. Non-initial identification process of MQT

Rounds	prefix	Response	Result	Q	CQ
				00,11,010,011, 101,1000,1001	
1	00	Idle		11,010,011, 101,1000,1001	00
2	11	1100,1101	Collision	010,011,101,1000, 1001,110,111	
3	010	Idle		011,101,1000, 1001,110,111	00,11,010
4	011	0110	Identified	101,1000, 1001,110,111	00,11, 010,011
5	101	Idle		1000,1001, 110,111	00,11,010, 011,101
6	1000	1000	Identified	1001,110,111	00,11,010, 011,101,1000
7	1001	Idle		110,111	00,11,010,011, 101,1000,1001
8	110	1100,1101	Collision	111,1100,1101	00,11,010,011, 101,1000,1001, 1100,1101
9	111	Idle		1100,1101	00,11,010,011, 101,1000,1001, 1100,1101,111
10	1100	1100	Identified	1101	00,11,010,011,101, 1000,1001,1100, 1101,111,1100
11	1101	1101	Identified		00,11,010,011,101, 1000,1001,1100,1101, 111,1100,1101

TABLE 4. Non-initial identification process of EBQT protocol

Rounds	Query string	Response	Updated string	Q	Identified tags
Initialization				00**,01**,10**,11**	
1	00**	Idle		01**,10**,11**	
2	01**	10	0110	10**,11**	0110
3	10**	00	1000	11**	1000
4	11**	110*	0101		1100,1101

Tables 3 and 4 compare MQT and EBQT in the non-initial process. From both the tables, we can see that EBQT outperforms MQT as well. The reasons are: (1) In MQT, when any tag leaves, the associated prefix makes an idle response (such as Round 7 in

Table 3) in the next process. Meanwhile, new arriving tags make new collisions (such as Round 8 in Table 3). However, in EBQT, the performance is related to only the number of tags, not relevant to the leaving and new arriving tags. Thus the performance is stable. (2) Each prefix can be regarded as a sub-tree. As aforementioned, EBQT not only avoids the idle responses, but also expands the prefix to the position of collided bits at once. Moreover, it automatically identifies two tags with only one bit collision, resulting in the quick identification with each prefix.

Beside the number of queries, EBQT requires the smaller number of bits (communication overhead) sent by tags and the smaller memory storage for prefixes (query strings). From Tables 1-4, we can get the following tables:

TABLE 5. Communication overhead of MQT and EBQT

	Initial process	Non-initial process
MQT	60 bits	24 bits
EBQT	30 bits	12 bits

TABLE 6. Memory requirement of MQT and EBQT

	Initial process	Non-initial process
MQT	21 bits (Q + CQ)	40 bits (Q + CQ)
EBQT	12 bits	16 bits

5. Mathematical Analysis.

We present some mathematical analyses for the performance of MQT and EBQT.

5.1. Queries of identification.

5.1.1. *MQT protocol.*

A. Initial identification process.

Lemma 5.1. *All tags to be identified by query strings are referred to the leaf nodes in a binary tree [13].*

Theorem 5.1. *Each non-leaf node represents a collision and requires two more queries to check its two child nodes.*

Proof: If a collision occurs when a reader sends query string $b_1b_2 \ldots b_x$, the reader expands the query to $b_1b_2 \ldots b_x0 \ldots$ or $b_1b_2 \ldots b_x1 \ldots$ to avoid collisions. This expansion continues until a single reply is received. There are three kinds of non-leaf nodes, as shown in Figure 9: three-degree (black nodes), two-degree (white nodes, such as node 1), and the root node. Note that the root node is separately handled from other two-degree nodes. The three-degree and root nodes need two queries to check their children. For any two-degree node, in addition to one query for its own child node, there is one zero reply problem (node 2), so it also needs two queries. Hence, in a binary tree, all non-leaf nodes represent collisions and each requires two queries to resolve the collision.

Lemma 5.2. *When N is the number of tags to be identified and $C(N)$ is the number of collisions in the query tree protocol, the number of queries that should be sent, $Qn(N)$, can be calculated as follows:*

$$Qn(N) = 2C(N) + 1$$

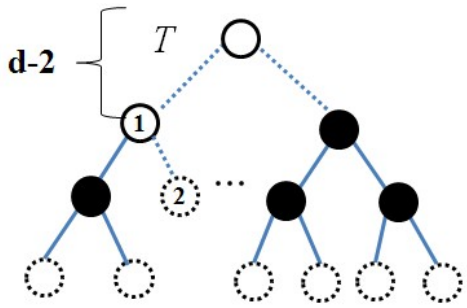

FIGURE 9. Tag identification process of query tree

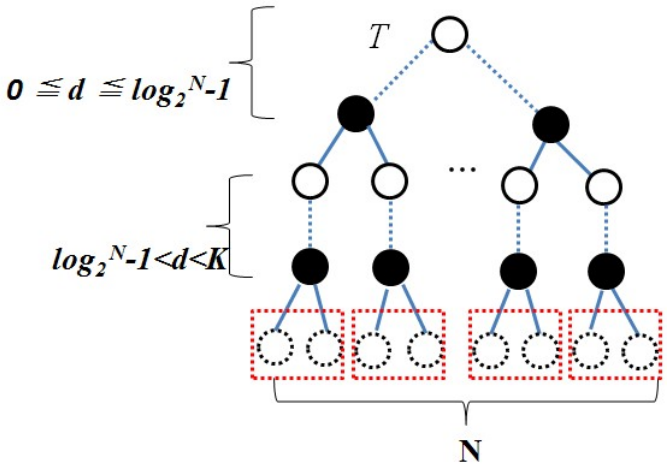

FIGURE 10. MQT protocol with the maximum collisions

Proof: By the proof of Theorem 5.1, all non-leaf node except the root node need two queries. The root node needs three queries, including the first default query.

Lemma 5.3. *For any set of N tags in the MQT protocol:*

$$2N - 1 \leq Qn(N) \leq N(K + 2 - \log_2 N) - 1$$

Proof: From Lemma 5.2, the worst-case for MQT is that the number of collisions is maximal. If each tag has a unique K bit ID, the number of collisions between two tags is maximally $K-1$, which is when the first $K-1$ bits of their IDs are the same. Thus, any two tags produce at most $K-1$ collisions. The maximum number of collisions in a tree with N tags and depth d, $C(N,d)$ can be acquired in the tree in Figure 10, where every two leaf nodes in dotted boxes experience $K-1$ collisions:

(a) When $0 \leq d \leq \log_2 N - 1$, the number of non-leaf nodes at the dth level is 2^d. This means that the maximum number of collisions $C(N,d)$ is 2^d.

(b) When $\log_2 N - 1 < d < K$, the number of non-leaf nodes at the dth level is $N/2$. This means that the maximum number of collisions $C(N,d)$ is $N/2$.

Based on (a) and (b), the number of collisions of the first $K-1$ bits is:

$$C(N)_{\text{MAX}} = \sum_{d=0}^{K-1} C(N,d) \leq \underbrace{(2^{\log_2^N} - 1)}_{(a)} + \underbrace{(K - \log_2^N)\frac{N}{2}}_{(b)}$$

$$= \frac{N}{2}(K + 2 - \log_2^N) - 1$$

Thus the maximum number of queries:

$$Qn(N)_{MAX} = 2C(N)_{MAX} + 1 = N(K + 2 - \log_2 N) - 1$$

Meanwhile, the best case performance for MQT can be achieved when the number of the collisions is minimal. Given N tags, the optimal binary tree is shown in Figure 11. All non-leaf nodes are confined within the first d levels and a collision happens at each node. Therefore, $C(N)_{MIN} = N - 1$, and the number of queries in the best-case is:

$$Qn(N)_{MIN} = 2C(N)_{MIN} + 1 = 2N - 1$$

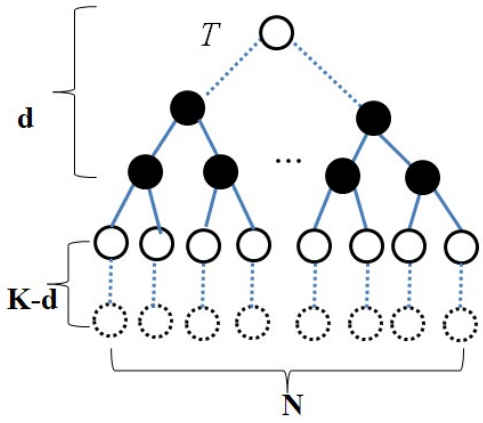

FIGURE 11. MQT protocol with minimum collisions

B. Non-initial identification process. In non-initial processes, let $Qn(P_i)$ be the set of tags recognized in the ith process and α and β be the number of arriving and leaving tags respectively. $Qn(P_{i+1}|P_i)$ denotes the number of queries transmitted by MQT to recognize tags in the $(i + 1)$th process. This substantially depends on the information taken from the ith process. Additionally, let N be the number of tags in the last process. Then we can obtain the following lemma:

Lemma 5.4. *When $\alpha = 0$ and $\beta = 0$, $Qn(P_{i+1}|P_i)$ can be calculated as follows:*

$$N \le Qn(P_{i+1}|P_i) \le N(K + 2 - \log_2 N)/2$$

Proof: $\alpha = 0$ and $\beta = 0$ mean that no tags leaving or arriving or no change in the set of tags. In the MQT protocol, the $(i + 1)$th process makes use of prefixes in CQ stored in the ith process. All the prefixes in CQ are either the ones which got one reply or idle prefixes in the previous process. As reported in [13], the maximum number of queries for identification is:

$$Qn(P_{i+1}|P_i)_{MAX} = N(K + 2 - \log_2 N)/2$$

If all of the leaf nodes represent tags not idle prefixes, each tag can be identified by one query. This is the best case for the number of queries for identification:

$$Qn(P_{i+1}|P_i)_{MIN} = N$$

Lemma 5.5. *When $\alpha > 0$ and $\beta = 0$, the number of queries is:*

$$N + \alpha \le Qn(P_{i+1}|P_i) \le N(K + 2 - \log_2 N) - (N - \alpha + 1)$$

Proof: $\alpha > 0$ and $\beta = 0$ indicate that some tags newly arrive without leaving tags. In the MQT protocol, idle prefixes in the previous process are used to identify the new tags in the next process [13]. If each new tag can be identified by an idle prefix, no collision occurs. In this case, the number of the identified and idle prefixes should be $N+\alpha$; hence, the best performance for identification is:

$$Qn(P_{i+1}|P_i)_{MIN} = N + \alpha$$

However, if any two tags share the same idle prefix, it results in a new collision. As mentioned in [13], the worst case for identification is:

$$Qn(P_{i+1}|P_i)_{MAX} = N(K + 2 - \log_2 N) - (N - \alpha + 1)$$

Lemma 5.6. *When $\alpha = 0$, $\beta > 0$, the number of queries for identification is:*

$$Qn(P_{i+1}|P_i) = N(K + 2 - \log_2 N)/2$$

Proof: $\alpha = 0$ and $\beta > 0$ imply that some tags leave without arriving tags. Because no tags arrive, no collision occurs. Some valid prefixes in the previous process become idle prefixes. Therefore, the number of queries for identification is fixed, being equal to the number of leaf nodes in the tree.

Lemma 5.7. *When $\alpha > 0$ and $\beta > 0$, the performance of time for identification is:*

$$N + \alpha \leq Qn(P_{i+1}|P_i) \leq N(K + 2 - \log_2 N) - (N - \alpha + 1)$$

Proof: $\alpha > 0$, $\beta > 0$ mean that some tags arrive while others leave. As mentioned in Lemma 5.6, the leaving tag does not increase the processing time for tag identification but the time is increased only by arriving tags. Instead, the leaving tags cause α idle prefixes.

$$Qn(P_{i+1}|P_i)_{MIN} = N + \alpha$$

The worst case is that all the arriving tags result in collisions, as analyzed in Lemma 5.5. The performance is:

$$Qn(P_{i+1}|P_i)_{MAX} = N(K + 2 - \log_2 N) - (N - \alpha + 1)$$

5.1.2. *EBQT protocol.*

A. Initial identification process. Unlike the existing QT protocols, the proposed EBQT knows at which bits collisions happen in each round. In next rounds, one of the collided bits of a query is replaced by '0' and '1' in order and sent again to indentify tags. The same value at the same bit does not cause a collision. Therefore, EBQT makes a full binary tree with minimum non-leaf nodes, as shown in Figure 12.

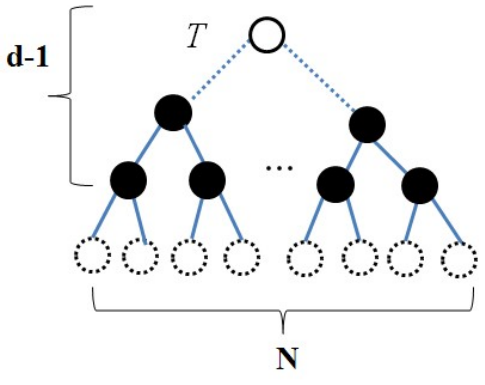

FIGURE 12. EBQT protocol with maximum collisions

Meanwhile, when only one '*' remains in a query string, two tags can be identified simultaneously without sending an extra query, resulting in time saving and communication overhead reducing. In this case, the leaf nodes in the last level can be pruned away, as shown in Figure 13.

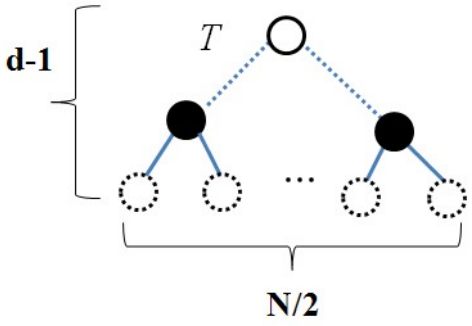

FIGURE 13. EBQT protocol with minimum collisions

Lemma 5.8. *For any set of N tags in EBQT protocol:*

$$N - 1 \leqq Qn(N) \leqq 2N - 1$$

Proof: Given N tags with K-bits IDs, the performance can be analyzed in two different situations:

(1) When $N = 2^K$, the binary tree is a complete binary tree, and the number of collisions is equal to the number of the non-leaf nodes.

$$Qn(N) = 2^K - 1 = N - 1$$

(2) When $N < 2^K$, the best case is that all N tags are leaf nodes in a complete tree. The number of collisions is equal to the number of leaf nodes.

$$Qn(N) = 2^K - 1 = N - 1$$

Meanwhile, in the worst case, each of N tags is divided into a different subset, thus the prefix has $\log_2 N$ bits difference. It is equivalent to a complete tree with $\log_2 N + 1$ level:

$$Q_n(N) = 2^{\log_2^N + 1} - 1 = 2N - 1$$

Conclusively, we can get:

$$N - 1 \leqq Qn(N) \leqq 2N - 1$$

B. Non-initial identification process. In non-initial processes, let $Qn(P_i)$ be the set of tags identified in the ith process and α and β be the number of arriving and leaving tags respectively. $Qn(P_{i+1}|P_i)$ denotes the number of queries transmitted by the EBQT in the $(i+1)$th process. When N is the number of identified tags in the previous process, we can obtain the following lemma:

Lemma 5.9. *When $\alpha = 0$ and $\beta = 0$, the number of queries for identification is:*

$$N/2 \leq Qn(P_{i+1}|P_i) \leq N$$

Proof: $\alpha = 0$ and $\beta = 0$ mean no change of tags as compared with the previous process. For the best case performance, N tags should be accurately divided into $N/2$ pairs. Then they can be identified with $N/2$ query messages. The worst performance is when each node needs an individual query, resulting in a total of N query messages.

Lemma 5.10. *When $\alpha > 0$ and $\beta = 0$, the number of queries for identification is:*

$$N/2 + (\alpha - 1) \leq Qn(P_{i+1}|P_i) \leq 2(N + \alpha) - 2$$

Proof: $\alpha > 0$ and $\beta = 0$ imply that some tags arrive without tags leaving. The number of queries for new tags depends on the number of them and total time for identification is confined in $[N - 1, 2N - 1]$ according to Lemma 5.8.

Lemma 5.11. *When $\alpha = 0$ and $\beta > 0$, the number of queries for identification is:*

$$N/2 \leq Qn(P_{i+1}|P_i) \leq N$$

Proof: $\alpha = 0$ and $\beta > 0$ present that some tags leaving without tags arriving. The leaving tags do not make collisions and not increase the delay for identification. So the performance is the same as the situation $\alpha = 0$ and $\beta = 0$.

Lemma 5.12. *When $\alpha > 0$ and $\beta > 0$, the number of queries for identification is:*

$$N/2 + (\alpha - 1) \leq Qn(P_{i+1}|P_i) \leq 2(N + \alpha) - 2$$

Proof: $\alpha > 0$ and $\beta > 0$ mean that some tags arrive while others leave. This is derived from Lemmas 5.10 and 5.11.

5.2. **Computing complexity.** The work of MQT and EBQT is composed of three operations in common:

TABLE 7. Operations in MQT and EBQT

Operations	Remark
Q	Query-Response
IS	Prefix insertion into queue
DE	Prefix deletion from queue

In the initial process, MQT needs the querying operation, Q_{MQT}, and prefixes inserting operations, IS_{MQT}. Note that $IS_{\mathrm{MQT}} > N$ because the prefixes to be inserted into CQ include N identified prefixes at least. Thus:

$$3N - 1 \leq Q_{\mathrm{MQT}} + IS_{\mathrm{MQT}} \leq N(K + 3 - \log_2 N) - 1$$

On the other hand, EBQT needs only the querying operation Q_{EBQT}, thus:

$$N - 1 \leq Q_{\mathrm{EBQT}} \leq 2N - 1$$

Compared with MQT, EBQT obviously has lower computing complexity in the initial processes.

In non-initial processes, MQT needs the complexity for not only Q_{MQT} and IS_{MQT}, but also DE_{MQT}. The maximum complexity for the prefixes deletion operation DE_{MQT} is $2N - 1$, then we can get:

$$4N + \alpha - 1 \leq Q_{\mathrm{MQT}} + IS_{\mathrm{MQT}} + DE_{\mathrm{MQT}} \leq N(K + 4 - \log_2 N) + \alpha - 2$$

On the other hand, in EBQT, it directly generates 2^{L-1} ($2^{L-1} \leq N$ and L is the starting level which mentioned in Section 4.1.b) prefixes and starts the querying operations IS_{EBQT}. The complexity for prefix generation is 2^{L-1}. According to the analysis above, we can get:

$$2N - 1 \leq Q_{\mathrm{EBQT}} + IS_{\mathrm{EBQT}} \leq 3N - 1$$

Thus, we can see that EBQT has lower computing complexity in both initial and non-initial process.

5.3. **Spatial complexity.** The memory requirement M to store ID prefixes is compared in this section.

In MQT, memory is demanded for two types of prefixes. One is temporary prefixes to track the identification process and the other is identified or idle prefixes. Thus the memory is needed as much as:

$$M_{\mathrm{MQT}} = [N(K + 2 - \log_2 N) - 1]K + (N + X_{\mathrm{idle}})K$$

here, X_{idle} is the number of idle prefixes.

On the other hand, EBQT only needs the memory to track the identification process, thus the maximum memory demand is:

$$M_{\mathrm{EBQT}} = (2N - 1)K$$

As $M_{\mathrm{MQT}} - M_{\mathrm{EBQT}} = K[N(K + 1 - \log_2 N) + X_{\mathrm{idle}}] > 0$, we can see that EBQT outperforms MQT in respect of spatial complexity.

6. **Performance Evaluation.** In this study, the tag ID is 32 bits long and randomly generated. Like all the previous works, the time delay to identify all tags is assumed to be proportional to the number of queries sent by the reader. The communication overhead is measured by the number of bits transmitted by tags. Performance of EBQT is compared with AIS, MQT, and BQT. Note that, in the static scenario, EBQT is the same as BQT.

6.1. **Static identification scenario.** In the static scenario, there is no leaving or arriving tag, and the performance was evaluated with different number of tags that are evenly deployed in the target area.

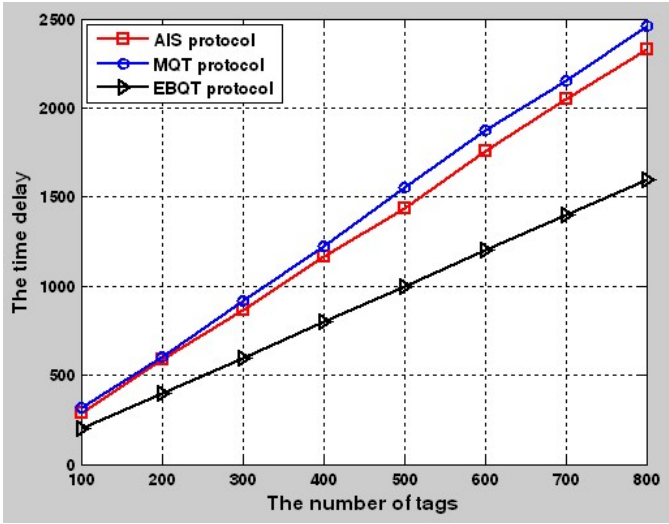

FIGURE 14. Time delay in static scenario

Figures 14 and 15 compare the time delay for identification and communication overhead against different numbers of tags in the static scenario (initial process). Figure 14 shows that EBQT outperforms the AIS and MQT protocols in respect of time delay. This is because the individual bit collision detection was introduced to reduce collisions. It does not only avoid the idle responses, but also accelerate the prefix expansion to reduce collisions. Unlikely, AIS makes use of only collision information in a left sub-tree to accelerate the identification process in a right sub-tree. Meanwhile, MQT has totally the same process as QT, thus it suffers from the worst performance.

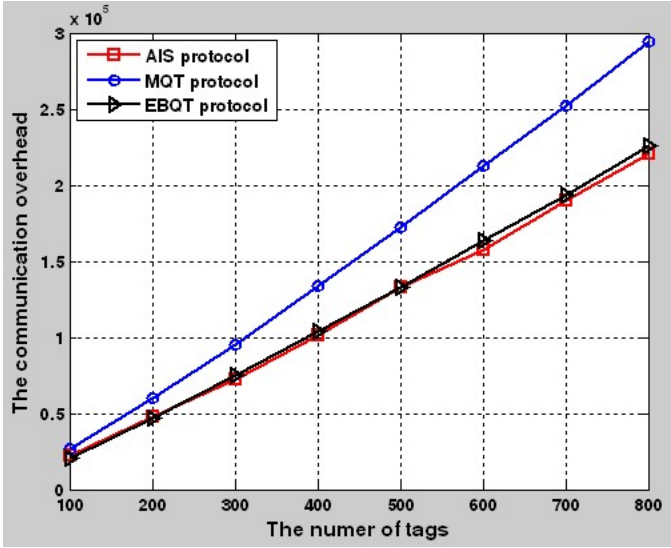

FIGURE 15. Communication overhead in static scenario

Due to the reduction of queries on tag identification, EBQT and AIS achieve better performance than MQT in respect of the communication overhead, as shown in Figure 15.

6.2. **Mobile identification scenario.**

6.2.1. *The impact of different number of tags.* First, performance was evaluated with the different numbers of tags that are evenly deployed in the target area under the same moving speed of a reader. Based on our observation on practical situations, the moving speed of a mobile reader keeps consistent and the changing rate of tags between two consecutive processes is fixed to 20%, as illustrated in Figure 16. ($d = 20\%$ means that in each process, there are 20% identified tags move out and 20% new tags move in).

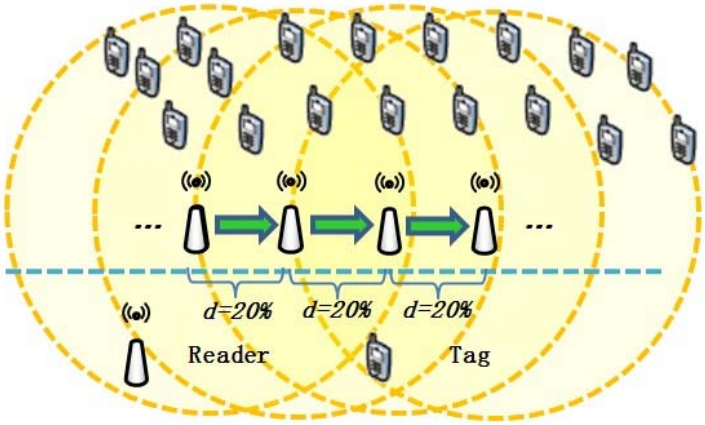

FIGURE 16. The scenario of fixed moving speed

Figures 17 and 18 compare the time delay for identification and communication overhead against different numbers of tags in a mobile scenario (non-initial process). Figure 17 shows that the EBQT protocol outperforms the others in respect of the number of queries. MQT merely employs relevant information between two consecutive processes to reduce the number of queries while BQT only adopts bit collision detection to expand

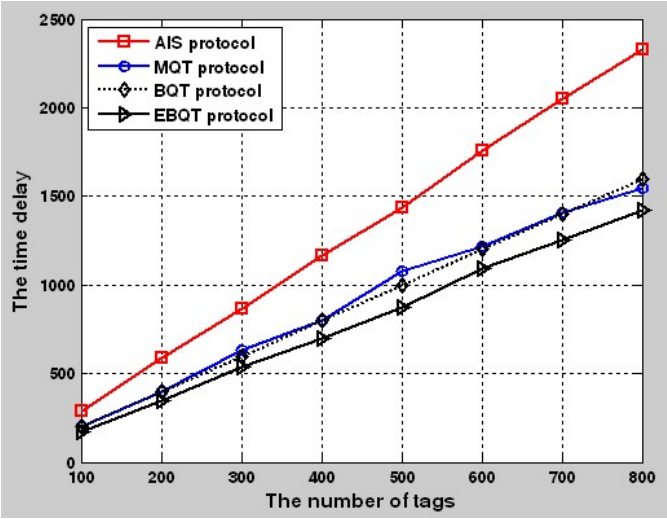

FIGURE 17. Time delay in mobile scenario (fixed moving speed)

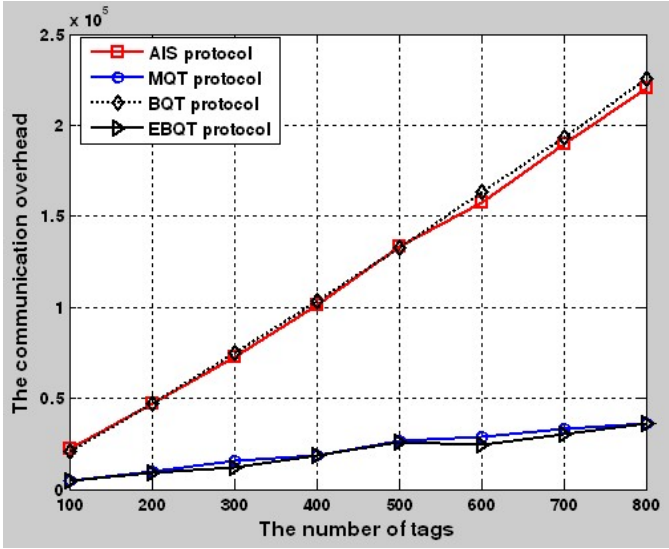

FIGURE 18. Communication overhead in mobile scenario (fixed moving speed)

prefix for collision reduction. AIS does not have any consideration of mobile identification. On the other hand, EBQT protocol does not only adopt the individual bit collision detection but also uses the relevant information to directly expand the prefixes to shorten identification process.

The decreased number of queries also reduces the communication overhead as shown in Figure 18.

6.2.2. *The impact of different speeds of mobile reader.* So far, we have assumed that a reader speed is fixed all the time, but the moving speed of a reader can be changed at any time, which means that the arrival and departure rates of tags may be different in each process. This is illustrated in Figure 19.

Figures 20 and 21 compare the time delay and communication overhead for identification against different moving speeds of a reader. The speeds are set up so that the rates of arriving and leaving tags may range from 10% to 100% in steps of 10 and the number of tags of each process is fixed to 800.

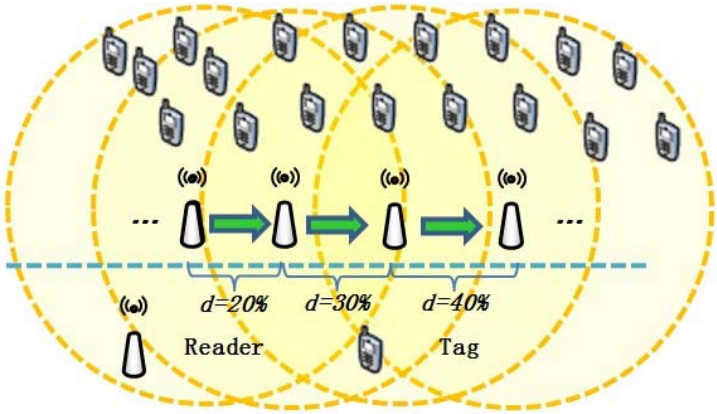

FIGURE 19. The scenario of different moving speed

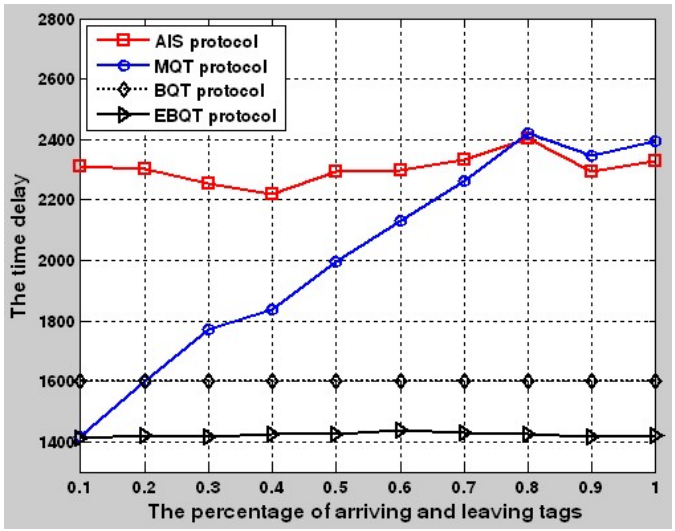

FIGURE 20. Time delay in mobile scenario (different moving speeds)

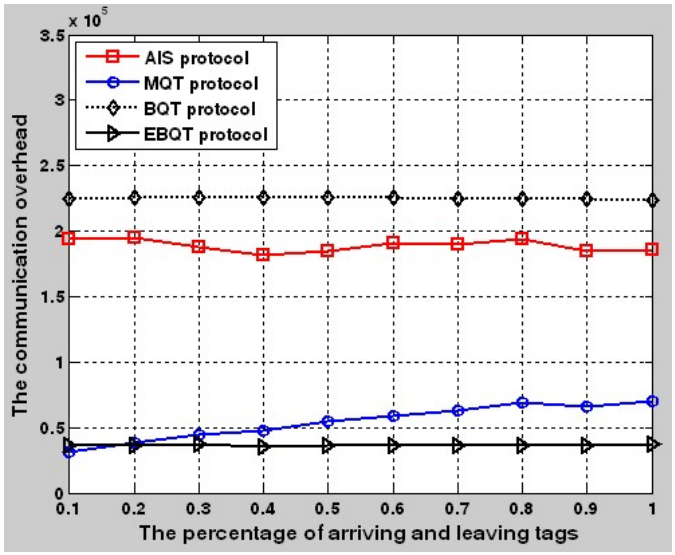

FIGURE 21. Communication overhead in mobile scenario (different moving speeds)

Figure 20 shows the EBQT protocol outperforms BQT, AIS and MQT all the time in respect of the number of queries. We can come to the following conclusions: (1) EBQT not only employs the collision reduction technology (bit collision detection) of BQT but also utilizes the relevant information like MQT to speed up the next identification process, thus BQT and MQT become the lower bound of the performance of EBQT. (2) Both MQT and EBQT utilize the relevant information to speed up the next identification process, but the performances under different moving speed are different. EBQT has very stable performance. The difference comes from the fact that MQT uses the prefixes generated in the previous identification process, but this information is useless when a lot of tags are changed between two consecutive processes. On the other hand, in EBQT, it generates the prefixes depending on the number of the tags achieved in the previous process. Thus its performance has nothing to do with the changing rate of tags. If the number of tags between two consecutive processes is similar, the performance keeps stable.

Due to the same reasons, in Figure 21, we also can see that EBQT has the best performance in respect of communication overhead, because of the decreased number of queries.

Meanwhile, thanks to the stable performance, it is easy to evaluate the identification latency and control the movement of the reader in our practical application. Moreover, with the moving speed increasing, the advantage of EBQT becomes more obvious. Thus we can shorten the whole identification time delay by increasing the moving speed, if there is no other potential problem.

7. **Conclusion.** In this paper, we improved our previous work BQT by employing the temporal and spatial correlation between two consecutive identifications. Due to the new improvement, our proposed EBQT achieve shorter identification time delay in both static and mobile identification scenario, and also makes the identification process be a very stable. Moreover, it does not increase the cost in other aspects. The analysis and simulation results show that EBQT can achieve a significant reduction in processing time and communication overhead for tag identification.

8. **Future Work.** As explained in Section 3, our EBQT adopts the Manchester code for individual bit collision detection. Thus the critical foundation of our protocols is the requirement of accurate time synchronization between all tags. A large time difference may degrade the performance of individual bit collision detection. In order to solve this problem, we are working on developing hardware and testing different modules to find an applicable way for individual bit collision detection with the tolerance for time differences.

Acknowledgment. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0006048). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] K. Finkenzeller, *RFID Handbook*, Carl Hanser Verlag, Munich, 2003.

[2] I.-C. Lin, C.-W. Yang and S.-C. Tsaur, Nonidentifiable RFID privacy protection with ownership transfer, *International Journal of Innovative Computing, Information and Control*, vol.6, no.5, pp.2341-2351, 2010.

[3] W.-T. Sung, J.-H. Chen, M.-H. Wang and Y.-C. Hsu, Remote medical care system design based on RFID and Zigbee technology via wireless sensors network, *International Journal of Innovative Computing, Information and Control*, vol.6, no.11, pp.5203-5220, 2010.

[4] J.-S. Chang and R.-S. Chang, A probabilistic cost estimation model for RFID real-time applications in data grids, *International Journal of Innovative Computing, Information and Control*, vol.6, no.12, pp.5421-5438, 2010.

[5] H.-Y. Chien and T.-C. Wu, Efficient and scalable key agreement schemes for mobile RFID readers and servers, *International Journal of Innovative Computing, Information and Control*, vol.6, no.12, pp.5463-5471, 2010.

[6] S. R. Lee, S. D. Joo and C. W. Lee, An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification, *Proc. of the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pp.166-174, 2005.

[7] J. R. Cha and J. H. Kim, Dynamic framed slotted ALOHA algorithms using fast tag estimation method for RFID system, *Proc. of the 3rd IEEE Consumer Communications and Networking Conference*, 2006.

[8] H. S. Choi, J. R. Cha and J. H. Kim, Fast wireless anticollision algorithm in ubiquitous ID system, *Proc. IEEE VTC 2004*, LA, USA, 2004.

[9] J. I. Capetanakis, Tree algorithms for packet broadcast channels, *IEEE Trans. Information Theory*, vol.25, pp.505-515, 1979.

[10] D. R. Hush and C. Wood, Analysis of tree algorithms for RFID arbitration, *Proc. IEEE Int'l Symposium on Information Theory*, pp.107, 1998.

[11] C. Law, K. Lee and K. Y. Siu, Efficient memoryless protocol for tag identification, *Proc. of the 4th Int'l Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pp.75-84, 2000.

[12] H. Vogt, Multiple object identification with passive RFID tags, *Proc. IEEE Int'l Conference on Systems, Man and Cybernetics*, vol.3, 2002.

[13] J. Myung, W. J. Lee and T. K. Shih, An adaptive memoryless protocol for RFID tag collision arbitration, *IEEE Transactions on Multimedia*, vol.8, no.5, 2006.

[14] C. H. Hsu, B. Y. Chen and C. T. Yang, Anticipative inquiry scheme for efficient RFID tag identification, *Proc. of IEEE 3rd Conference on Multimedia and Ubiquitous Engineering*, 2009.

[15] K. W. Chiang, C. Q. Hua and T. S. Peter, Prefix-randomized query tree protocol for RFID system, *Proc. IEEE ICC*, 2006.

[16] J. Ryu, H. J. Lee, Y. H. Seok, T. Y. kwon and Y. H. Choi, A hybrid query tree protocol for tag collision arbitration in RFID system, *Proc. IEEE ICC*, 2007.

[17] J. H. Choi, D. W. Lee and H. J. Lee, Query tree-based reservation for efficient RFID tag anti-collision, *IEEE Communication Letters*, vol.11, no.1, 2007.

[18] J. Ma and X. Wei, An improved anti-collision algorithm in RFID system, *Proc. of the 9th Int'l Conference on Hybrid Intelligent System*, 2009.

[19] L. Liu, Z. Xie, J. Xi and S. Lai, An improved anti-collision algorithm in RFID system, *Proc. of the 2nd International Conference on Mobile Technology, Applications and Systems*, 2005.

[20] S. S. Kim, Y. H. Kim, S. J. Lee and K. S. Ahn, An improved anti-collision algorithm using parity bit in RFID system, *Proc. of the 2nd International Conference on Mobile Technology, Applications and Systems*, 2005.

[21] H. S. Gou, H. C. Jeong and Y. H. Yoo, A bit collision detection based query tree protocol for anti-collision in RFID system, *Proc. of WiMob*, 2010.