A New Adaptive Query Tree on Resolving RFID Tag Collision

Ching-Nung Yang, *Member, IEEE*, Yu-Ching Kun, Chih-Yang Chiu, and Yu-Ying Chu

***Abstract*—A so-called tag-collision problem (or missed reads) in Radio Frequency Identification (RFID) system is the event that a reader cannot identify the tag if many tags respond to a reader at the same time. In this paper, we propose an adaptive query tree (AQT) protocols that combine the multiple *M*-ary trees. The large *M* reduces collisions, while the small *M* avoids the unnecessary inquiries. Our AQT protocol successfully decreases the identification time by reducing the collisions and the idle time simultaneously. Two variant modes were proposed in our AQT, the base AQT (BAQT) and the modified AQT (MAQT), designed for the random electronic product code (EPC) and the very similar EPC with the first half bits are identical. Simulation results reveal that the proposed BAQT and MAQT outperform the base query tree.**

I. Introduction

RADIO frequency identification (RFID) system consists of radio tags, readers and the backend database system. Readers communicate with tags at distance through wireless transmission. The reader interfaces with the backend system to identify a radio tag. RFID tag works as a wireless barcode, and can hold more data than the bar codes [1]. Many manufacturers see the intended applications of RFID technology, and deploy RFID in inventory control, distribution industry, and supply chain management [2, 3]. As far, there are already many concrete applications using RFID. However, a so-called collision problem degrades the tag identification efficiency. Such unreliable identification will compromise the usefulness of RFID system. There are two types of collisions- the reader collision and the tag collision [4]. Reader collision is that two or more readers scan a tag simultaneously [5]. On the other hand, the tag collision is the identification problem when a reader receives more than one tag at the same time [6].

Several technologies available on tag collision had been proposed. These anti-collision protocols are often categorized into ALOHA-based protocol and Tree-based protocol. ALOHA-based protocol [7] reduces the tag collisions, while it has the starvation problem (a tag cannot be identified for a long time). To solve such situation, two tree-based protocols- the binary tree (BT) and the query tree (QT)- were accordingly proposed. In BT protocols [8-11], every tag generates a random number (0 or 1). Afterwards, the tags having "0" transmit their Electronic Product Codes (EPCs) to the reader, and the tags having "1" transmit later. A reader repeats this procedure until all tags are identified.

This work was supported in part by Testbed@TWISC, National Science Council under the Grants NSC 98-2219-E-006-001.

C.N. Yang is with the National Dong Hwa University, #1, Da Hsueh Rd, Sec. 2, Hualien, Taiwan (corresponding author to provide phone: 886-3-8634025; fax: 886-3-8634010; e-mail: cnyang@mail.ndhu.edu.tw).

978-1-4244-6700-6/10/$26.00 © 2010 IEEE

QT protocol does not need the additional memory and is so referred to as the memory-less protocol. A reader in QT protocol sends a prefix of EPC to query the tags. Tags matching the prefix will then respond to a reader. Extend the prefixes until only one tag's ID matches, i.e., a tag can be identified successfully. In this paper, we propose a new QT-based identification algorithm to enhance the identification efficiency. Some QT-like protocols are briefly introduced in the following. In the prefix-randomized QT (PQT) [12], a reader first scans the neighboring tags to determine an *M*-ary tree for querying tags. After finishing the *M*-ary tree, a reader then uses binary tree for inquiries. The adaptive query splitting (AQS) protocol in [13] used extra candidate queue to store the prefix bits of responded tags to speed up the identification process. A hybrid QT (HQT) in [14], the fixed 4-ary QT and the slotted back-off tag response mechanism were used to avoid collision. In [15], two *M*-ary trees were combined to form a unified QT (UQT), and enhanced the identification efficiency. In the enhanced QT (EQT) [16], the length of prefix code is adjusted dynamically according to the length of tag ID. All QT-like protocols in [12-16] could achieve the reliable and efficient identification.

In this paper, we design two adaptive QT (AQT) protocols by combining the multiple *M*-ary (M=2^1, 2^2, 2^3, …) trees. Our AQT protocols enhance the identification efficiency. The rest of this paper is organized as follows. A brief introduction of QT is shown in Section II. In Section III, we propose two AQT algorithms for tag anti-collision. Performance and comparison are given in Section IV, and we draw our conclusion in Section V.

II. Preliminaries

Two QT-like protocols are described. One is the base QT (BQT) and the other is *M*-ary tree based QT (MQT). It is evident that MQT is reduced to BQT for M=2. Some notations used in this paper are first defined in Table I.

TABLE I Notation Used

t	the tag's ID (e.g. EPC), a y-tuple t=(t_1, t_2, …, t_y), where t_i∈{0, 1}, 1≤i≤y, and y is the number of bits of tag's ID; for example EPC has y=96
q	the query strings sent from a reader , the x-tuple q=(q_1, q_2, …, q_x), where q_i∈{0, 1}, 1≤i≤x, and x(≤y) is the length of a query string
Q	a queue maintained by a reader, which stores the query strings
Push (Q, s)	push a query bit string s into the queue Q
Pop (Q)	pop a query string q=Pop(Q) from the queue Q
M(q, t)	verify the query string q whether matches the prefix of tag's ID t, i.e., M(q, t) checks whether

	$q \overset{?}{=} (t_1, t_2, \ldots, t_{\lvert q \rvert})$, and if $q = (t_1, t_2, \ldots, t_{\lvert q \rvert})$, then $M(q, t)$=1 else $M(q, t)$=0
M-ary tree	the *M*-ary tree, where M=2^i, $1 \le i \le b$
n	the number of total tags for identification
n_t	the number of collided tags for a query
R_i	the threshold range defines which *M*-ary (M=2^i) tree should be used in our AQT protocol by checking $n_t \in R_i$=[min_i, max_i]

A. BQT Anti-Collision Protocol

In BQT, a reader send a query string q to ask tags whether their IDs contain a prefix $(t_1, t_2, \ldots, t_{\lvert q \rvert})$ same to q. If two or more tags answer (i.e. a collision is detected at communication channel), there are two or more tags having the same prefix. The reader then appends bit 0 and 1 to form the longer prefixes (q0) and (q1) to query tags. Repeat the query procedure until all tags are uniquely identified. Fig. 1 illustrates this BQT algorithm.

Example 1. Suppose the length of tag's ID is 6 bits (i.e., y=6), and all t=(t_1, t_2, t_3, t_4, t_5, t_6) of 9 tags are {(000000), (010011), (010010) (010001), (011000), (011110), (110010), (110100), (110111)}. At first, a reader broadcasts 1-bit query string q=0, six tags will respond, and a collision is detected since there are 6 tags having t_1=0. By using Push (Q, 00) and Push (Q, 01), a reader adds two query strings into queue Q={1, 00, 01}. On the other hand, there is a collision when a reader sends q=1 (since 3 tags have t_1=1), and then Q is updated as={00, 01, 10, 11}. Finally, there are total 23 interrogation cycles, which includes 10 collision cycles, 4 idle cycles and 9 successful cycles. □

```
Algorithm : BQT
/* initialize Q */
if (Q = NULL)
    Push(Q, 0);
    Push(Q, 1);
end
begin
  while (Q!=NULL)
    q=Pop(Q);
    reader broadcast q to tags;   /* reader query tags*/
    switch(M(q, t))   /*tag compare its ID with the query q*/
      case (M(q, t)=1):
        if (collision)      /* two or more tags respond */
          Push(Q, q0);
          Push(Q, q1);
        else                /* only one tag responds */
          tag transmit its ID t to reader;
        end
      break;
      case (M(q, t)=0):
        no tag responds;   /* idle cycle */
      break;
  end while
end
```

Fig. 1. BQT algorithm.

B. MQT Anti-Collision Protocol

MQT protocol is based on *M*-ary tree (M=2^b) and is an extension of BQT. It reduces the collisions, but has more idle cycles. The MQT algorithm is shown in Fig. 2. In MQT, if a collision occurs, a reader will append b-tuples to form M longer prefixes $(q\overbrace{0\cdots00}^{b})$, $(q\overbrace{0\cdots01}^{b})$, …, and $(q\overbrace{1\cdots11}^{b})$. Repeat the matching and querying procedure until all tags are uniquely identified.

Example 2. Consider nine tags in Example 1, and use MQT algorithm with M=4 (i.e. b=2) to identify them. MQT reduces the collisions from 10 to 5, while the number of idle cycles is increased from 4 to 7. The overall interrogation cycles are 5(collision)+7(idle)+(successful)=21. When compared with BQT, 4-ary QT save us 2 interrogation cycles. □

```
Algorithm : MQT
/* initialize Q */
if (Q = NULL)
    Push (Q, $\overbrace{0\cdots00}^{b}$); …; Push (Q, $\overbrace{1\cdots11}^{b}$);   /* add M query strings, where M=$2^b$ */
end
begin
  while (Q!=NULL)
    q=Pop(Q);
    reader broadcast q to tags;   /* reader query tags*/
    switch(M(q, t))   /*tag compare its ID with the query q*/
      case (M(q, t)=1):
        if (collision)      /* two or more tags respond */
          Push (Q, q$\overbrace{0\cdots00}^{b}$); …; Push (Q, q$\overbrace{1\cdots11}^{b}$);   /* add M query strings, where M=$2^b$ */
        else                /* only one tag responds */
          tag transmit its ID t to reader;
        end
      break;
      case (M(q, t)=0):
        no tag responds;   /* idle cycle */
      break;
  end while
end
```

Fig. 2. MQT algorithm.

Example 3. Consider nine tags in Example 1, and use MQT algorithm with M=8 (i.e. b=3) to identify them. There are 4 collisions; however the number of idle cycles is increased up to 20. The overall interrogation cycles are 4(collision)+ 20(idle)+9(successful)=33. In this example, the value of M is too large to efficiently identify 9 tags. □

III. The Proposed AQT Protocols

A. BQT Anti-Collision Protocol

It is observed that applying a fixed *M*-ary tree in MQT is not an effective solution to enhance BQT. Example 2 and Example 3 reveal that the large *M*-ary tree reduces the collisions but increase the unnecessary inquiries. Also, Example 3 implies that the choice of M is critical, and should be suitably chosen for the different numbers of responding tags. From the preceding description and the results in Examples, unfortunately, there exists a dilemma of using the large *M*-ary tree and the small *M*-ary tree. When studying how to unify the multiple *M*-ary trees to enhance the identification efficiency, we first carefully observe their distinguishing characteristics.

The large *M*-ary tree:

Advantage: Using the large *M*-ary tree implies the longer prefix, so that the tags may have the same prefix with the small probability. This small matching probability reduces the tag's collisions.

Disadvantage: The less matching of the query string and the tag's ID increases the invalid inquiries.

The small *M*-ary tree:

Advantage: The size of a query prefix is lesser than that of the large *M*-ary tree, and thus using the small *M*-ary tree will have the more valid inquiries. Therefore, the query has a less idle time.

Disadvantage: Tags may have the high possibility holding the same short prefix, and this will considerably increases the collisions.

In the followings, we give a theoretical analysis for the above properties of the large M-ary tree and the small M-ary tree. Two probabilities are defined- the average probability of collisions P_C and the average probability of idle cycles P_I for n tags using MQT. Let U_t^j be the event that some other tags have the same j-bit prefix of the tag t, which the j-bit prefix is the j-bit query string q sent by the reader, and V_t^j be the event that no tag has the same j-bit prefix as the query string q (i.e., $M(q, t)$=0).

We first derive the probability of U_t^j as follows.

$$\begin{aligned}\text{Prob}\left(U_t^j\right) &= \text{Prob}\{\text{some other tag ID has the same } (t_1,t_2,\ldots,t_j)\}\\ &= 1-\text{Prob}\{\text{all other tag IDs do not have } (t_1,t_2,\ldots,t_j)\}\\ &= 1-\text{Prob}\{\text{an ID does not have } (t_1,t_2,\ldots,t_j)\}^{n-1}\\ &= 1-\left(1-\left(1/2^b\right)^j\right)^{n-1}. \text{ (note: the reader uses } 2^b\text{-ary tree for query)}\end{aligned} \tag{1}$$

Then, the average probabilities of collisions P_C for 96-bit EPC is then determined in the following.

$$P_C=\left(\sum_{j=1}^{(96/b)} 1-\left(1-(1/2^b)^j\right)^{n-1}\right)\Big/(96/b). \tag{2}$$

Lemma 1. The probability P_C using 2^{b_1}-ary tree is lesser than the collision probability P_C using 2^{b_2}-ary tree for b_1> b_2.

Proof. Without loss of generality, we simply suppose b_1=(a×b_2), where a>1, and 96 are both divisible for both b_1 and b_2. Let P_C^1 and P_C^2 be the probability P_C using 2^{b_1}-ary tree and 2^{b_2}-ary tree, respectively.

$$\begin{aligned}P_C^2 &= \left(\sum_{j=1}^{(96/b_2)} 1-\left(1-\left(1/2^b\right)^j\right)^{n-1}\right)\Big/(96/b_2)\\ &= \Big(\underbrace{1-\left(1-1/2^{b_2}\right)^{n-1}+\cdots+1-\left(1-1/2^{ab_2}\right)^{n-1}}_{}+\\ &\underbrace{1-\left(1-1/2^{ab_2+1}\right)^{n-1}+\cdots+1-\left(1-1/2^{2ab_2}\right)^{n-1}}_{}+\cdots+\\ &\underbrace{1-\left(1-1/2^{(96/ab_2)\times ab_2-ab_2+1}\right)^{n-1}+\cdots+1-\left(1-1/2^{(96/ab_2)\times ab_2}\right)^{n-1}}_{}\Big)\Big/(96/b_2)\\ &> \Big(\underbrace{1-\left(1-1/2^{ab_2}\right)^{n-1}+\cdots+1-\left(1-1/2^{ab_2}\right)^{n-1}}_{}+\\ &\underbrace{1-\left(1-1/2^{2ab_2}\right)^{n-1}+\cdots+1-\left(1-1/2^{2ab_2}\right)^{n-1}}_{}+\cdots+\\ &\underbrace{1-\left(1-1/2^{(96/ab_2)\times ab_2}\right)^{n-1}+\cdots+1-\left(1-1/2^{(96/ab_2)\times ab_2}\right)^{n-1}}_{}\Big)\Big/(96/b_2)\\ &= \Big(\underbrace{1-\left(1-1/2^{ab_2}\right)^{n-1}}_{}+\underbrace{1-\left(1-1/2^{2ab_2}\right)^{n-1}}_{}+\cdots+\underbrace{1-\left(1-1/2^{(96/ab_2)\times ab_2}\right)^{n-1}}_{}\Big)\Big/(96/ab\\ &= \Big(\underbrace{1-\left(1-1/2^{b_1}\right)^{n-1}}_{}+\underbrace{1-\left(1-1/2^{2b_1}\right)^{n-1}}_{}+\cdots+\underbrace{1-\left(1-1/2^{(96/b_1)\times b_1}\right)^{n-1}}_{}\Big)\Big/(96/b_1)\\ &= P_C^1.\end{aligned} \tag{3}$$

From (3), we have $P_C^1 < P_C^2$ for b_1>b_2. The proof is completed. □

We then derive the probability of V_t^j as follows.

$$\begin{aligned}\text{Prob}\left(V_t^j\right) &= \text{Prob}\{\text{no tag has the same } j\text{-bit prefix as the query string}\}\\ &= \text{Prob}\{\text{all tag IDs do not have } (t_1,t_2,\ldots,t_j)\}\\ &= \text{Prob}\{\text{an ID does not have } (t_1,t_2,\ldots,t_j)\}^n\\ &= \left(1-\left(1/2^b\right)^j\right)^n.\end{aligned} \tag{4}$$

Then, the average probabilities of idle cases P_I for EPC 96 bits is then determined in the following.

$$P_I=\left(\sum_{j=1}^{(96/b)}\left(1-(1/2^b)^j\right)^n\right)\Big/(96/b). \tag{5}$$

Lemma 2. The probability P_I using 2^{b_1}-ary tree is lesser than the probability P_I using 2^{b_2}-ary tree for b_1< b_2.

Proof. Let P_I^1 and P_I^2 be the idle probability P_I using 2^{b_1}-ary tree and 2^{b_2}-ary tree, respectively. By the same approach in the proof of Lemma 1, we can prove $P_I^1 < P_I^2$ for b_1< b_2. We omit the complete proof for brevity. □

Lemma 1 and Lemma 2 demonstrate the opposite characteristics for the large *M*-ary tree and the small *M*-ary tree- the large *M*-ary tree reduces the collision, and the small *M*-ary tree avoids the unnecessary inquiries. In Fig. 3(a), there are 11 collided nodes and 4 idle nodes when executing BQT protocol. However, when using 8-ary tree, the number of the collisions in this identification procedure is significantly reduced from 11 to 3, while the number of idle nodes increases to 14 (see Fig. 3(b)). The results consist with Lemma 1 and Lemma 2.

In order to bring these conflicting goals (the reduction of collision and the less idle time) in a kind of balance, we properly combine the different *M*-ary trees in our AQT protocols to develop their specialities and simultaneously avoid the corresponding disadvantages. We tries to reduce the identification time by reducing the collisions and the idle time simultaneously, and this delivers the following problems: (1) what are the appropriate *M*-ary trees required in our AQT protocols? (2) when do we use the different *M*-ary trees querying tags in our AQT protocols? (3) how do we reduce the number of idle cycles for the large *M*-ary tree?

By try and error, the first two problems can be easily solved. About the third problem, we need an extra approach to delete the unnecessary inquiries. To avoid the unnecessary inquiries, if we had queried all the tags in a collided node, we may skip the remainder leaves of this collided node. This approach is only effective for the idle nodes at the tail leaves. For example, as shown in Fig. 3(b), 9 idle nodes marked by a dotted line can be discarded. By this discarding approach, the number of idle nodes is reduced from 14 to 5. A simulation result discarding the idle cycles in MQT (M=2, 8, 32) for testing 2000 tags is shown in Fig. 4. This approach really works, and can reduce the number of idle cycles.

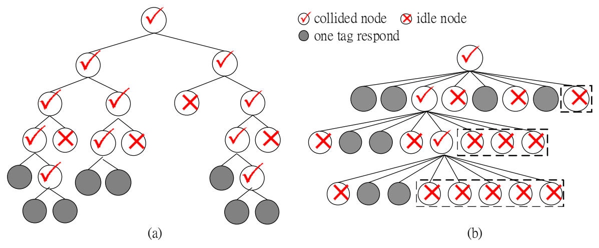

Fig. 3. Identification of eight tags using the fixed MOT: (a) BOT (b)

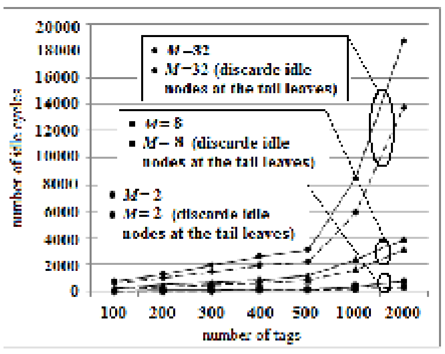

Fig. 4. The improvement for MQT with M=2, 4 and 8 by discarding the idle nodes at the tail leaves.

B. Base Adaptive Query Tree (BAQT)

The BAQT algorithm is shown in Fig. 5. In the initial phase, we first set the threshold ranges, R_1, R_2, …, R_b, and add M ($M=2^i$, $i\in[1, b]$) query string with the *i*-tuples if the number of total tags for identification n belongs the R_i region ($n\in R_i$). When the collision occurs, we use M-ary ($M=2^i$, $i\in[1, b]$) tree for identification by checking the number of collided tags n_t belongs the R_i ($n_t\in R_i$). Also, we skip the remainder leaves of this collided node to avoid the unnecessary inquiries if we had queried all the tags in a collided node.

```
Algorithm : BAQT
/* initialize $Q$ */
if (Q = NULL)
    Set $R_i$, $i\in[1, b]$;
    if ($n\in R_i$) {Push ($Q$, $\overset{i}{0\cdots00}$); …; Push ($Q$, $\overset{i}{1\cdots11}$);};   /* add M query strings, where $M=2^i$ */
end
begin
  while (Q!=NULL)
    if (the remainder number of tags in a collided node =0) skip the remainder leaves of this collided node;
    $q$=Pop($Q$);
    reader broadcast $q$ to tags;   /* reader query tags*/
    switch ($M(q, t)$)   /*tag compare its ID with the query $q$*/
      case ($M(q, t)$=1):
        if (collision)       /* two or more tags respond */
          if ($n_t\in R_i$) {Push ($Q$, $q\overset{i}{0\cdots00}$); …; Push ($Q$, $q\overset{i}{1\cdots11}$);};   /* add M query strings, where $M=2^i$ */
        else                 /* only one tag responds */
          tag transmit its ID $t$ to reader;
        end
      break;
      case ($M(q, t)$=0):
        no tag responds;   /* idle cycle */
      break;
  end while
end
```

Fig. 5. BAQT algorithm.

Example 4. Consider nine tags in Example 1, and use our BAQT algorithm with M-ary trees of M=2, 4, 8. Also, the threshold ranges we used for this example are R_1=[1, 2], R_2=[3, 4], R_3=[5, 9]. At first, a reader detects n=9 tags for identification; since $n\in R_3$ therefore an 8-ary tree is used. When broadcast the query string (000), a tag with ID (000000) responds. At the collided nodes, 010, 011, 110, there are, respectively n_t=3, 2 and 3 collided tags. The remainder number of collided nodes for the original root is n_1=9−1−3−2−3=0, and thus the query string (111) can be discarded. Since the numbers of collided tags at the nodes (010) and (110) are $n_t=3\in R_2$, both collided nodes use 4-ary trees. So, the query strings (01000), (01001), (01010), (01011), and (11000), (11001), (11010), (11011) are put into the queue Q. On the other hand, the value of n_t in (011) node is $2\in R_1$, so the query strings (0110), (0111) are added into the queue Q. At the collided node (010), a reader finishes the querying (01000) by receiving the unique response from the tag with ID (010001). When querying (01001), a reader detects two collided tags from (010010) and (010011). Therefore the remainder number of collided nodes for the (010) root is n_2=3−1−2=0, so the querying strings (01010) and (01011) can be discarded. Repeat checking the remainder number of collided tags in a node, we could avoid the unnecessary inquiries. Table II lists the detail of identifying procedure. There are 5 collision cycles, and 4 idle cycles. The overall interrogation cycles are 5(collision)+4(idle)+ 9(successful)=18 is better than MQT with M=4 (21 interrogation cycles), MQT with M=8 (33 interrogation cycles) and BQT (23 interrogation cycles). □

C. Modified Adaptive Query Tree (MAQT)

In [15], the authors show a possible application scenario of RFID, a so-called warehouse distribution. It is reasonable to assume that the EPC data of most items from the same warehouse will be very similar since the items are manufactured by the same company, and are stacked together in a large warehouse. As we know, EPC has 96 bits embracing four sections- the header (H: 8 bits), the GMN (G: 28 bits), the object class (O: 24 bits), and the serial number (S: 36 bits). These items with the similar IDs may have the same encoding scheme (header), the same company prefixes (GMN), the same object class, and the different serial Number. Each EPC can be simply divided into two parts the first half bits are identical, and the second half where most bits are unique. Let the lengths of the first half and the second half be l_f and l_s where $l_f+l_s=96$. The tag-collision problem in the warehouse distribution scenario often has the identical $l_f=60$ bits (H+G+O) and the different $l_s=36$ bits (S). In [15], two other possible scenarios were also shown- the same $l_f=36$ bits (H+G) and the different $l_s=60$ bits (O+S); the same $l_f=8$ bits (H) and the different $l_s=88$ bits (G+O+S).

The following lemma shows that the same prefixes of tags result in a significantly increasing number of idle nodes in an *M*-ary tree.

Lemma 3. When identifying the tags with the same l_f-bit prefixes by MQT (where $M=2^b$), the number of idle cycles is $(2^b-1)\times(l_f/b)$ for querying the first l_f bits.
Proof. Consider the case using 2^b –ary tree to identify the tags having the same l_f bits. We will have and (l_f/b) layers to finish querying the first l_f bits, and have (2^b-1) idle nodes in each layer. Thus, the number of idle nodes is $(2^b-1)\times(l_f/b)$. □

Lemma 3 reveals that the number $(2^b-1)\times(l_f/b)$ exponentially increases when the value of *b* increases. Suppose $l_f=60$ (i.e., the same header, the same GMN, the same object class), then $(2^b-1)\times(l_f/b)=60$ and 630 for $b=1$ and 6, respectively. Because the deletion of idle nodes in BAQT only works at the tail leaves, the number of remaining idle nodes is still considerable for the large *b*.

If we identify the tags with the very similar EPC by using the large *M*-ary trees, the same prefixes will cause the large idle cycles. Since using the prefixes for identification, the performance is very sensitive to the distribution of tags' EPCs To address the problem of the very similar EPC, we propose the MAQT protocol that combines BQT and BAQT. We use BQT protocol (the smallest *M*, i.e., $M=2$) when the length of query string is no larger than l_f (i.e., $|q|\le l_f$), in order to reduce the impact of increasing idle cycles. On the other hand, the BAQT is adopted for $l_f\le|q|\le l_f+l_s$. Fig. 6 illustrates the MAQT algorithm.

```
Algorithm : MAQT
/* initialize $Q$ */
if ($Q$ = NULL)
    Set $R_i$, $i \in [1, b]$;
    Push($Q$, 0);
    Push($Q$, 1);
end
begin
    while ($Q$!=NULL)
        If ( $|q| \le l_f$ )    /* if the length of query string is no larger than the prescribed
                                   length of the first half of EPC then using BQT protocol */
            do BQT protocol;
        else
            do BAQT protocol;  /* the length of query string is larger than the prescribed
                                  length of the first half of EPC then using BAQT protocol */
        end
    end while
end
```

Fig. 6. MAQT algorithm.

IV. PERFORMANCE EVALUATION

Two experiments are conducted to evaluate the performance of BQT, MQT, and the proposed BAQT and MAQT. Experiment #1 uses the test tags with the random EPC, while Experiment #2 uses the tags with the very similar EPC. In both experiments, we take n=100, 200, 300, 400, 500, 1000 and 2000 (the number of total tags) for test. The goal of experiments is to evaluate the identification performance, and measure the sensitivity of EPC. Also, we will discuss the results of the experiments. Two problems mentioned in section III: (1) what are the appropriate *M*-ary trees required in our AQT protocols? (2) when do we use the different *M*-ary trees querying tags in our AQT protocols? By the rule of thumb, the answers have been given as R_1=[1, 3], R_2=[4, 8], R_3=[9, 15], R_4=[16, 30], R_5=[31, 60], R_6=[61, 125], R_7=[126, 250], R_8=[251, 500], R_9=[501, 1000], $R_{10}=[1001, \infty]$.

In our BAQT and MAQT, ten threshold ranges, R_1–R_{10} (it means that we use 10 *M*-ary trees: 2-ary tree, 4-ary-tree, …, 1024-ary tree) are used. For example, when the number of collided tags in a query is $n_t=58$, we use 32-ary tree in our protocol since $n_t \in R_5$.

Experiment #1. Four protocols are tested: the BQT, the MQT with M=16, the proposed BAQT, and the proposed MAQT. The EPC data of test tags are randomly generated, and the length is 96 bits.

We randomly generate *n* tags' IDs (96-bit EPC). Apply BQT, MQT, BAQT and MAQT, respectively, to identify *n* (100, 200, 300, 400, 500, 1000 and 2000) tags. For each *n*, we repeat the same experiment five times to calculate the average number of collision cycles N_C, the average number of idle cycles N_I, and the average number of total interrogation cycles N_T. Table III summarizes the test results. It is observed that MQT has the least collisions of all four protocols but significantly increases the idle cycles; finally, it has the most interrogation cycles. Our two protocols outperform BQT and MQT. BAQT and MAQT have the almost same idle cycles. However, BAQT has the lesser collisions than MAQT, so that it has the least interrogation cycles. For example, for n=2000, the average numbers of total interrogation cycles N_T for BAQT, MAQT, BOT and MQT are, respectively, 4112, 4762, 5522 and 9772. Our BAQT has the best performance among four protocols. □

Experiment #2. Redo Experiment #1, but use test tags with the very similar EPC data. In this experiment, all tags have the same l_f=60 bits and the random l_s=36 bits.

TABLE III

N_C, N_I AND N_T WHEN USING THE TAGS WITH RANDOM EPC

	BQT			**MQT**			**BAQT**			**MAQT**		
n	N_C	N_I	N_T	N_C	N_I	N_T	N_C	N_I	N_T	N_C	N_I	N_T
100	140	42	282	30	369	499	74	33	207	100	22	222
200	287	89	576	70	869	1139	158	73	431	247	46	503
300	430	132	862	110	1306	1716	214	136	650	370	66	736
400	548	150	1098	150	1878	2428	295	120	815	488	78	966
500	703	205	1408	190	2366	3056	410	135	1045	633	99	1232
1000	1399	401	2800	350	4278	5628	809	274	2083	1159	204	2363
2000	2760	762	5522	609	7169	9772	1589	523	4112	2380	382	4762

We first choose a 60-bit prefix, and then randomly generate 36 bits. The values of N_C, N_I and N_T are shown in Table IV. It is observed that the number of total interrogation cycles N_T of BQT, MQT and BAQT protocols increase due to the same prefixes. Our MAQT, respectively, uses BQT in the first l_f bits and BAQT in the last l_s bits to achieve a better performance. For example, for n=2000, the average number of total interrogation cycles N_T for BAQT, MAQT, BOT and MQT are, respectively, 7256, 4326, 5786 and 10505. The N_I of BAQT significantly increases from 523 to 3549 for n=2000 (as mentioned in Lemma 3), while MAQT only has N_I=605 (since it uses BQT in the first l_f bits). Due to the increasing of N_I, BAQT has the more N_T than BQT and MAQT. Our MAQT has the least N_T among four protocols. The approach combining BQT and BAQT is really effective for identifying the tags with the very similar EPC data. □

V. Conclusion

In this paper, we proposed the new adaptive query tree protocols to resolve the tag-collision problem in RFID network. By unifying multiple M-ary trees together and discarding the idle nodes at the tail leaves, our adaptive approaches achieve the least total interrogation cycles. Two variant modes of our adaptive query tree protocols are proposed, BAQT and MQAT, which address the different distribution of tags' EPCs. Experimental results show that BAQT and MAQT have the best performance for the tags with random EPC and the tags with the very similar EPC.

TABLE IV

N_C, N_I AND N_T WHEN USING THE TAGS WITH THE VERY SIMILAR EPC

	BQT			**MQT**			**BAQT**			**MAQT**		
n	N_C	N_I	N_T	N_C	N_I	N_T	N_C	N_I	N_T	N_C	N_I	N_T
100	210	112	422	49	657	806	98	373	571	148	71	319
200	345	147	692	84	1085	1369	169	687	1056	218	96	514
300	489	192	981	126	1609	2025	238	1151	1689	266	166	732
400	628	230	1258	168	2141	2710	325	1115	1840	371	162	933
500	780	282	1562	212	2696	3408	376	1294	2170	483	177	1160
1000	1497	499	2996	375	4650	6025	759	2639	4398	909	340	2249
2000	2892	894	5786	655	7850	10505	1667	3549	7256	1721	605	4326

References

[1] F. Klaus, RFID Handbook: *Fundamentals and Applications in Contactless Smart Cards and Identification*, John Wiley & SonS, 2003.

[1] M. L. Chuang, M. H. Shaw, "RFID: Integration stages in supply chain management," *Engineering Management Review*, vol. 35, pp. 80-87, 2007.

[2] Y. Li, and X. Ding, "Protecting RFID communications in supply chains," in *Proc. 2nd ACM symposium on Information, computer and communications security*, Singapore, 2007, pp. 234–241.

[3] S. Sarma, S. Weis, and D. Engels, "RFID systems and security and privacy implications," *LNCS*, vol. 2523, pp. 454–470, 2003.

[4] J. Waldrop, and D. Engels, and S. Sarma, "Colorwave: an anticollision algorithm for the reader collision," in *Proc. IEEE International Conf. Communication,* Louisiana, 2003, pp. 1206–1210.

[5] C. Floerkemeier, M. Lampe, "Issues with RFID Usage in Ubiquitous Computing Applications," *LNCS*, vol. 3001, pp. 188–193, 2004.

[6] S.R. Lee, S.D. Joo, C.W. Lee, "An Enhanced Dynamic Framed Slotted ALOHA Algorithm for RFID Tag Identification," in *proc. The 2nd Annual International Conf. on Mobile and Ubiquitous Systems: Networking and Services*, San Diego, 2005, pp. 166–172.

[7] F. Zhou, D. Jin, C. Huang, and H. Min, "Optimize the power consumption of passive electronic tags for anti-collision schemes," *The 5th International Conference on ASIC*, vol. 2, pp.1213–1217, 2003.

[8] B. Feng, J.T. Li, J.B. Guo, and Z.H. Ding, "Id-binary tree stack anti-collision algorithm for RFID," in *proc.11th IEEE Symposium on Computers and Communications*, Sardinia, 2006, pp. 207–212.

[9] J. Myung, W. Lee, "Adaptive binary splitting: a RFID tag collision arbitration protocol for tag identification," *Mobile Networks and Applications*, vol. 1, pp. 347–355, 2006.

[10] Y. Cui, Y. Zhao, "Mathematical Analysis for Binary Tree Algorithm in RFID" in Proc. 67th IEEE Vehicular Technology Conf., Singapore, 2008, pp. 2725–2729.

[11] K.W Chiang, C. Hua, T.S. Yum, "Prefix-Randomized Query-Tree Protocol for RFID System," *IEEE International Conference on Communication*, vol. 4, pp. 1653–1657, 2006.

[12] J. Myung, W. Lee, J. Srivastava, T.K. Shih, "Tag-splitting: adaptive collision arbitration protocols for RFID tag identification," *IEEE Trans. On Parallel and Distributed Systems*, vol. 18, pp.763–775, 2007.

[13] J. Ryu, H. Lee, Y. Seok, T. Kwon, Y. Chioi, "A hybrid query tree protocol for tag collision arbitration in RFID system," in *Proc. IEEE International Conf. on Communications*, Glasgow, 2007, pp. 5981-5986.

[14] P. Pupunwiwat, B.Stantic, "Unified q-ary tree for RFID tag anti-collision resolution," in *Proc.20th Australasian Database Conference*, Wellington, 2009, pp. 49–58.

[15] C.H. Hsu, C.H. Yu, Y.P. Huang, "An enhanced query tree (EQT) protocol for memoryless tag anti-collision in RFID system," in *Proc. 2nd International Conf. on Future Generation Communication and Networking*, Hainan Island, 2008, pp. 427–432.