Four-ary Query Tree Splitting with Parallel Responses for RFID Tag Anti-collision

Ming-Kuei Yeh[*] Jehn-Ruey Jiang[†] Shing-Tsaan Huang[†]
jamesyeh@webmail.ntcb.edu.tw jrjiang@csie.ncu.edu.tw sthuang@csie.ncu.edu.tw

[†]Department of Computer Science and Information Engineering
National Central University
No.300, Jhongda Rd., Jhongli City, Taoyuan, 32001, Taiwan

[*]Department of Information Management
National Taipei College of Business
No.321, Sec. 1, Jinan Rd., Zhongzheng District, Taipei City 100, Taiwan

Corresponding Author: Prof. Jehn-Ruey Jiang
Telephone: +8863-422-7151 ext 35312
Fax: +8863-422-2681
E-mail: jrjiang@csie.ncu.edu.tw
Postal address: No. 300, Jhongda Rd., Jhongli City, Taoyuan, 32001, Taiwan

ABSTRACT

In an RFID system, tags can be identified by a reader when they are within the interrogation zone of the reader. Collisions occur when tags using backscatter technology report their IDs to the reader at the same carrier frequency simultaneously. The performance of tag identification is degraded by collisions, so it is important to address the tag collision problem. Several anti-collision protocols have been proposed for reducing tag collisions. They can be categorized into two classes: ALOHA-based and tree-based protocols. This paper proposes a 4-ary query tree-based anti-collision protocol, namely the parallel response query tree (PRQT) protocol, to improve the performance of tag identification. In the PRQT protocol, the tags with the ID prefix matching either the reader's request bit string or the complementary of the string are arranged to respond in two subcarriers in parallel. The PRQT protocol is analyzed, simulated, and compared with related ones to demonstrate its advantages.

Keywords: RFID, anti-collision, query tree, tag identification

1. INTRODUCTION

The front end of an RFID (**R**adio **F**requency **ID**entification) system is composed of readers and tags [1]. When a tag and a reader are close enough, they can communicate with each other. For such a situation, it is said that the tag is in the *interrogation zone* of the reader. In most cases, the reader does not know which tags are in its interrogation zone. It must initiate an *interrogation procedure* (or *identification procedure*) to request tags to report their IDs. When multiple tags respond to the reader request simultaneously, signal interference occurs and no tag can be identified by the reader successful. How to reduce tag collisions to speed up the interrogation procedure is thus important. Several *anti-collision* protocols are proposed to reduce tag collisions. They can be categorized into two classes: ALOHA-based and tree-based protocols.

In the ALOHA-based protocols [2,3], tags respond to the reader by transmitting IDs in a probabilistic manner. For example, in the slotted ALOHA protocol [3], the whole span of an interrogation procedure is divided evenly into several time slots, and a tag randomly chooses a time slot for transmitting its ID to the reader at the beginning of that time slot. The ALOHA-based protocols are simple; however, they have the *tag starvation problem* that a tag might not ever be identified successfully since its responses always collide with those of others.

The basic idea of the tree-based protocols [4-16] is to repeatedly split the group of tags encountering collisions into subgroups until there is only one tag in a subgroup to be identified without collisions. The tree-based protocols can be further categorized into *stateful* and *stateless* protocols [7]. In the stateful protocols [5,6,10], the tag needs to memorize the reader's inquiry state and reacts according to the state, so it needs on-tag memory and more complex circuits of higher costs. On the contrary, in the stateless protocols [7-9, 11], the tag responds to the reader's request only according to the current reader's inquiry information. It needs no memory to keep the on-going inquiry states, leading to simpler tag circuits of lower costs. This paper thus focuses on the stateless protocols, especially the well known query tree (QT) protocol and its variants [9, 13-16].

In this paper, a novel stateless query tree-based anti-collision protocol is proposed, called the *parallel response query tree (PRQT)* protocol, based on the concepts of *parallel prefix matching* and *parallel subcarrier responding* to improve the identification performance. In the PRQT protocol, a reader issues a request ~~bit~~ string *S* to tags, and the tags with ID prefixes matching *S* or the complement of *S* can respond to the reader's request in two subcarriers in parallel. To be more precise, by the parallel prefix matching scheme, a tag performs the exclusive-or (XOR) operation on each bit of *S* with the tag ID MSB (most significant bit). If the tag has the ID prefix matching the XOR result, it will respond to the request by reporting its ID postfix. With the help of the parallel subcarrier responding scheme, the tag response is transmitted in either of two subcarriers according to the XOR of the first bit of the ID postfix and the tag ID MSB. The PRQT protocol is analyzed, simulated, and compared with related ones to demonstrate its advantages.

The rest of this paper is organized as follows. The QT protocol and its variants are introduced in Section 2. The PRQT protocol is proposed in Section 3 and analyzed in Section 4. The protocol is simulated and compared with related ones in Section 5. And finally, conclusions are drawn in Section 6.

2. RELATED WORK

In this section, the QT protocol and its variants, which are stateless anti-collision protocols most related to the proposed PRQT protocol, are introduced in this section. Specifically, the QT, BSQTA, BSCTTA, AQS, CTTA and M-ary tree protocols are described below.

2.1 The QT Protocol

In the QT protocol [7], a reader starts a *round* of tag identification to identify all tags by sending the *request bit string* (or *request string* for short) S="0" to tags and pushing the request string "1" into the empty stack of the reader. A tag with tag ID prefix matching S will respond with its whole tag ID to the reader. If only one tag responds at the meantime, the tag is identified successfully and the reader pops up from the stack the next request string to execute the next tag interrogation. But if multiple tags respond simultaneously, collisions occur. In such a case, the reader sends S0 (S appended with 0) to all tags and pushes S1 (S appended with 1) into the stack. This is equivalent to split the tags with prefix S into two subgroups of tags with prefixes S0 and S1. The above-mentioned procedure will be performed repeatedly to identify every tag in the interrogation zone until the stack is empty. The reader can then reset S and the stack to initiate another round of tag identification. It is noted that the tag-group splitting can be illustrated as a binary tree, called the *interrogation tree* or *identification tree*.

An example [16] of the tag identification process of the QT protocol is shown below. It is assumed that there are 8 tags with unique IDs, 0000100, 0010100, 0011010, 0011101, 1010110, 1011000, 1100111 and 1101110, respectively. The process is described iteration by iteration, where an *iteration* is for a reader to send a command and for tags to perform associated actions.

(1) The reader sends out a request string S="0" first and pushes the request string "1" into the stack. The tags with IDs 0000100, 0010100, 0011010 and 0011101 have the first bit of tag ID matching the request string S. They respond their tag IDs to the reader simultaneously and collisions occur.

(2) The reader then sends out a bit longer request string *S*="00" and pushes "01" into the stack. The tags with IDs 0000100, 0010100, 0011010 and 0011101 respond their tag IDs to the reader simultaneously and collisions occur again.

(3) The reader then sends out a bit longer request string *S*="000" and pushes "001" into the stack. In this case, only the tag with ID 0000100 responds the request and is identified successfully.

(4) For the case of successful identification, the reader pops "001" from the stack and sends it out as a request string. The tags with IDs 0010100, 0011010 and 0011101 respond their tag IDs to the reader simultaneously and collisions occur

(5) The reader then sends out a bit longer request string *S*="0010" and pushes "0011" into the stack. Only the tag with ID 0010100 responds the request and is identified successfully.

(6) For the case of successful identification, the reader pops "0011" from the stack and sends it out as a request string. The tags with IDs 0011010 and 0011101 respond their tag IDs to the reader simultaneously and collisions occur.

(7) The reader then sends out a bit longer request string *S*="00110" and pushes "00111" into the stack. Only the tag with ID 0011010 responds the request and is identified successfully.

(8) For the case of successful identification, the reader pops "00111" from the stack and sends it out as a request string. Only the tag with ID 0011101 responds the request and is identified successfully.

(9) For the case of successful identification, the reader pops "01" from the stack and sends it out as a request string. None of the tags has an ID prefix matching the request string *S*, so no response is received by the reader.

(10) For the case of no response, the reader pops "1" from the stack and sends it out as a request string. The tags with IDs 1010110, 1011000, 1100111 and 1101110 respond the request simultaneously and collisions occur again.

The identification procedure is executed repeatedly until the stack is empty, and then all tags can be identified successfully. By the example shown above, it can be observed that the QT protocol's identification delay is affected by the length and the distribution of tag IDs. Specifically, if the tags have continuous tag IDs, the request string *S* will grow longer and longer and the delay time of the identification procedure will then increase significantly.

2.2 The BSQTA and the BSCTTA Protocols

Choi et al. [13] observed that in the QT protocol the tag with ID prefix matching the request string *S* of length *k* will respond with its partial tag ID from the $(k+1)^{\text{th}}$ to the n^{th} (i.e., the last) bits to the reader when the reader sends to tags the request string *S*. If collisions happen, the reader needs to send the request string *S*0 and *S*1 to tags latter. But the request string *S*0 and *S*1 are only different at the last bit. On the basis of this observation, two methods, namely the bi-slotted query tree algorithm (BSQTA) and the bi-slotted collision tracking tree algorithm (BSCTTA), are proposed to reduce the identification time by using two consecutive response time slots. Below, the procedures of the two algorithms (or protocols) are introduced step by step.

(1) After several repeated identification, the reader pops a request string *S* of length *h*-1 from the stack and sends it to tags.

(2) The tag in the interrogation zone of the reader will respond with its partial tag ID to the reader in one of two consecutive time slots if the first *h*-1 bits of the tag ID matches with *S*. If the h^{th} bit of the ID is '0', the tag responds in the first response time slot; otherwise, it responds in the second one.

- For BSQTA, the tag responds with its partial ID from the $(h+1)^{th}$ bit to the last bit.
- For BSCTTA, the tag responds with its partial ID from the $(h+1)^{th}$ bit to the last bit until it receives an ACK command, which is sent by the reader to indicate collision occurrence.

(3) If there is no collision in a time slot, the responding tag can then be identified successfully.

(4) If collisions occur in a response time slot (numbered with 0 or 1), then the reader should send a new longer request string to tags.

- For BSQTA, the new request string will be *S* appended by the time slot number (0 or 1).
- For BSCTTA, the new request string will be *S* appended by the bits received before the first bit collision occurs.

The above procedure is repeated until all tags are identified successfully. As shown in [13], the performance of the QT protocol can be improved significantly by BSQTA and BSCTTA.

2.3 The AQS Protocol

The AQS (Adaptive Query Splitting) [14] protocol is proposed to improve the QT protocol on the basis of tag ID information obtained from the last identification round. It is suitable for the situation that the tag population does not change greatly in consecutive rounds. The identification procedure of the AQS protocol is the same as that of the QT protocol except that the request strings in the ready-to-send string queue is copied from the last identification round at the beginning of a new round. The queue includes not only the request strings of successful tag identification but also those without any tag response. If the population of tags in the interrogation zone remains the same, all tags can be identified successfully without modifying any request string in the queue. But if there is any tag joining or leaving after the last identification round, the following actions must be done.

- Tag joining:

If tag collisions occur for the request string *S* provided by the last identification round, there must be a new tag moving into the interrogation zone of the reader after the last identification round. For such a case, the tree splitting procedure is performed and longer request strings are added into the ready-to-send queue.

- Tag leaving:

  If some tag leaves, there will be no response for some request string *S* provided by the last identification round. In order to improve the identification performance, the reader should merge such a request string *S* with the one in the ready-to-send queue that has the same bit string as *S* except for the last bit.

2.4 The CTTA Protocol

The collision tracking tree algorithm (CTTA) [15] assumes two technologies to reduce the number of reader requests in the QT protocol. The two technologies are the bit collision detection and the full duplex transmission. With the bit collision detection technology, the reader can clearly detect which bits of the tag IDs sent by tags collide. With the full duplex transmission technology, a tag can respond its tag ID and receive the command sent from the reader simultaneously. Below, the tag identification procedure of CTTA is described step by step.

(1) When the reader pops a *k*-bit request string *S* from the stack and sends it to tags, a tag with its ID prefix matching *S* will respond with its tag ID from the $(k+1)^{th}$ bit to the last bit to the reader.

(2) During the period of receiving the reminder tag IDs responded by tags, the reader will check each bit and broadcast a signal to inform tags to stop responding when the first ID bit collision occurs (say, at the $(k+m)^{th}$ bit). In this way, the time that tags use to respond with the IDs from the $(k+m+1)^{th}$ bit to the last bit can be saved, since the colliding tags need to re-respond with their tag IDs.

(3) If there is no bit collision, the tag ID can be identified. However, if a collision first occurs at the $(k+m)^{th}$ bit, the reader pushes two new request strings, the ID prefix of $(k+m-1)$ bits appended with "0" and the ID prefix of $(k+m-1)$ bits appended with "1", into the stack. It is different from the QT protocol in which the reader just pushes the ID prefix of *k* bits appended with "0" and the ID prefix of *k* bits appended with "1" into the stack. It is obvious that the number of request strings sent by the reader can be reduced and the identification performance can be improved significantly.

(4) If the stack is empty, the identification procedure is finished; otherwise, the reader loops back to step (1) and pops a new request string from the stack.

In order to detect the collision at each separate bit correctly, timing synchronization among tags is very critical [16] for CTTA and its variants [18-21]. However, achieving accurate time synchronization among nearby tags is very challenging. Also, it is impossible for low cost tags, such as those conforming to the EPCglobal C1 G2 standard [22], to transmit tag IDs and detect collision signals simultaneously [23].

2.5 The M-ary Tree Protocol

The M-ary Tree (M-Tree) protocol [16] tries to split the group of colliding tags into M (M>2), instead of 2 in the QT protocol, sub-groups at a time by appending $\lceil \lg \mathrm{M} \rceil$ bits to the request string *S*. In the M-Tree protocol, the number of iterations with tag collisions decreases with M and the number of iterations with no tag response increases with M. To achieve a compromise between the two numbers is thus important for the M-Tree protocol. As shown in the paper [16], the identification performance of the M-Tree protocol is optimal when M = 3. The paper [9] proposes the hybrid query tree (HQT) protocol using the 4-ary tree and the slotted backoff mechanism to improve the tag identification procedure. On receiving the request string *S* of length *k* from the reader, the tags choose one of the four backoff time slots (numbered with 0, 1, 2 and 3) to respond their partial tag IDs according to the $(k+1)^{\text{th}}$ and $(k+2)^{\text{th}}$ bits of their own tag IDs. If there is only a tag responding in a time slot, the tag can be identified successfully. However, if there are multiple tags responding in the x^{th} time slot, where x = 0, 1, 2 or 3, then the reader can infer the occurrence of collisions and it appends to S the two bits 00, 01, 10 or 11 of value *x* for further tag identification. With the help of the slotted backoff mechanism, the number of iterations of tag identification can be reduced significantly.

3. THE PROPOSED PROTOCOL

The proposed PRQT protocol adopts the concept of the M-Tree protocol to have a 4-ary interrogation tree instead of a 2-ary (binary) interrogation tree in the original QT protocol. As described in Section II, the 4-ary query tree has the advantage that it causes fewer iterations with collisions than the 2-ary query tree. However, the former also causes more idle iterations (i.e., iterations with no tag response) than the latter. The PRQT protocol tries to reduce the number of collisions and idle iterations simultaneously to improve the identification performance by using two schemes, *parallel prefix matching* and *parallel subcarrier responding*, which will be described below.

3.1 Parallel Prefix Matching

The PRQT protocol adopts the tag circuit shown in Fig. 1 to achieve parallel prefix matching. By the circuit, when a tag receives the request string *S* sent from the reader, it performs the exclusive-or (XOR) operation on each received bit with the tag ID MSB (most significant bit). In that way, the tags whose ID prefix matches *S* or $\bar{S}$ (the complement of *S*) will assume a match occurs and respond to the reader request simultaneously. For example, if the request string *S* is 010 and there are two tags with IDs 010011 and

101110 in the reader interrogation zone, the tag with ID 010011 will regard the request string as S="010", while the tag with ID 101110 will regard the request string as $\bar{S}$="101". Either tag will assume that its tag ID prefix matches the request string and sends the tag ID remainder (011 or 110) to the reader. It is efficient for tags whose ID prefixes match S or $\bar{S}$ to send their ID remainders to the reader simultaneously. However, a mechanism is needed to make the tag ID remainders sent in parallel and be received properly by the reader. Such a mechanism using the concept of parallel subcarrier responding is introduced below.

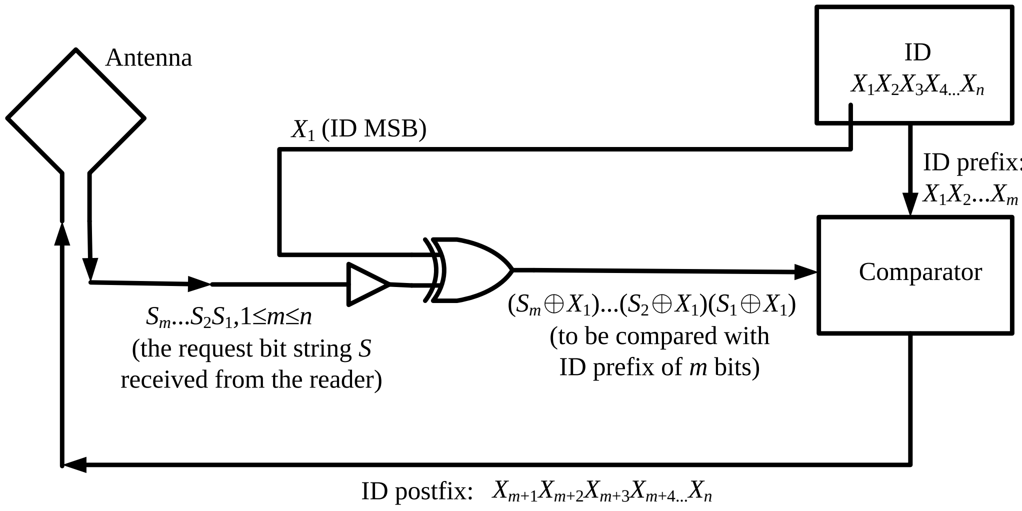

Antenna
ID
$X_1X_2X_3X_4...X_n$
X_1 (ID MSB)
ID prefix:
$X_1X_2...X_m$
Comparator
$S_m...S_2S_1, 1 \le m \le n$
(the request bit string S received from the reader)
$(S_m \oplus X_1)...(S_2 \oplus X_1)(S_1 \oplus X_1)$
(to be compared with ID prefix of m bits)
ID postfix: $X_{m+1}X_{m+2}X_{m+3}X_{m+4}...X_n$

Figure 1. The tag circuit design for parallel prefix matching

3.2 Parallel Subcarrier Responding

Since it is hard for the tag to perform complex signal transmission, the proposed parallel subcarrier responding scheme simply combines the FSK (Frequency Shift Keying) modulation and Manchester encoding techniques to provide two subcarriers for tags to send signals in parallel. With the FSK modulation technique [24], two sub-carrier tones in 2.2 MHz and 3.3 MHz [10] based on the baseband carrier in 900MHz can be used by two different tags to backscatter in parallel two separate signals to be received properly by the reader at the same time. The reader just uses the low-pass filter to filter out the baseband backscatter carrier because the tag responds on the wave of baseband backscatter, as shown in the right part of Fig. 2. It then separates the two subcarrier tones into subcarrier 0 and subcarrier 1 by band-pass filters.

The backscatter bits are encoded as Manchester codes, in which a low-to-high transition stands for 0, and a high-to-low transition stands for 1. With such an encoding mechanism, the collisions can be detected by the reader when two or more tags respond concurrently in either subcarrier. For the purpose of having the same bit transmission time, a bit is represented by 2 *basic transmission units*, either of which is composed of 4 (resp., 6) low and 4 (resp., 6) high consecutive amplitude waves in subcarrier 0 (resp., 1), as shown in the left portion of Fig. 2. The whole signal representation is shown in Fig. 3.

The tag whose ID prefix matches the request string S or complement of S will respond with its ID postfix or remainder R starting from the $(|S|+1)^{th}$ bit to the last bit. If the result of the XOR operation on the ID MSB and the first bit of R (namely R_1) is 0, then R will be sent in subcarrier 0; otherwise, in subcarrier 1.

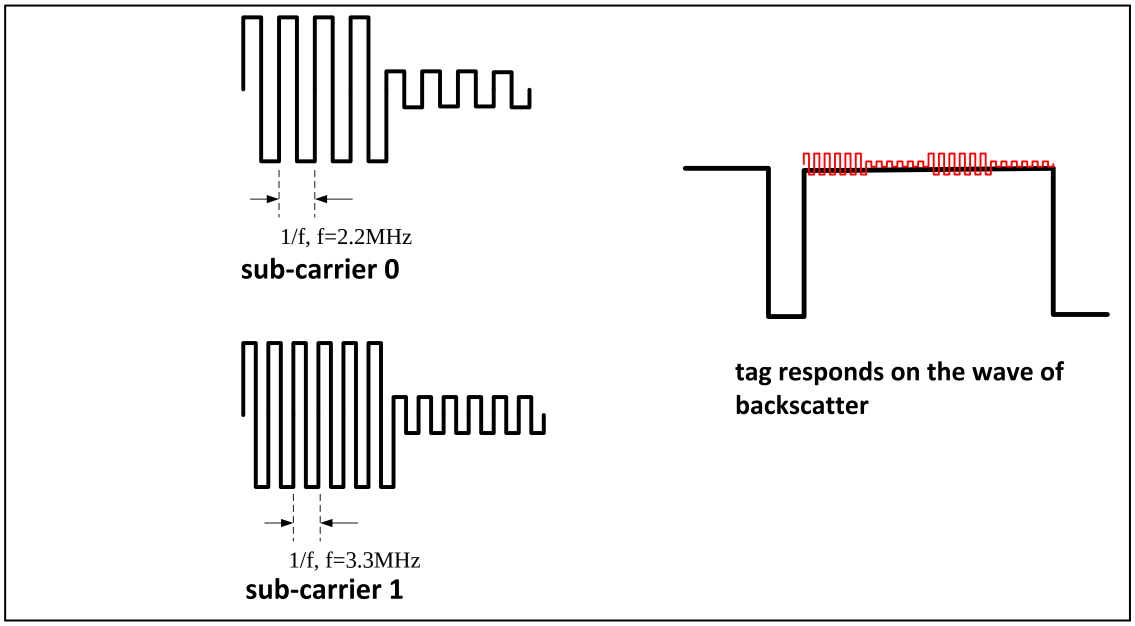


Figure 2. The data encoding and modulation for parallel subcarrier responding

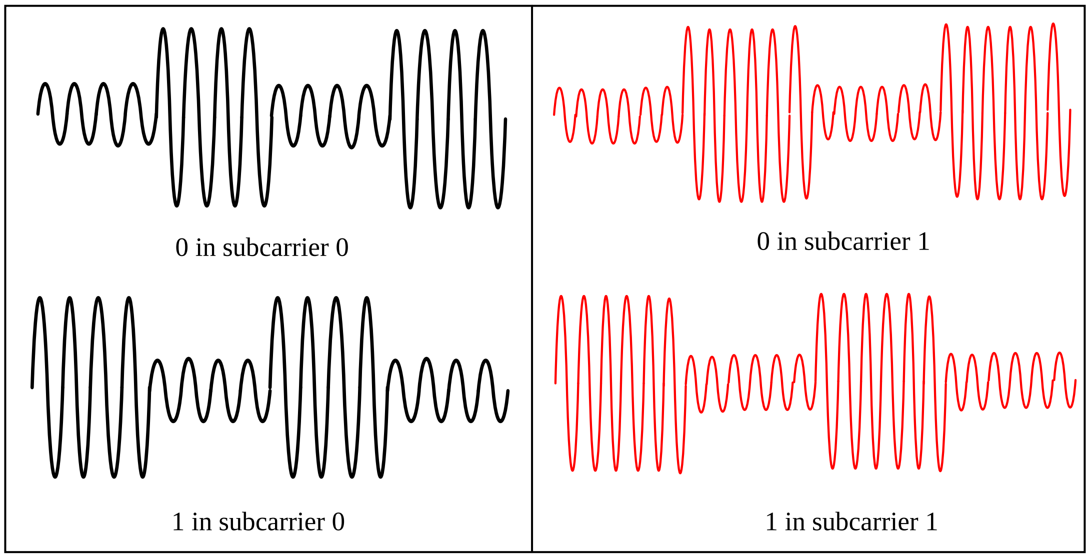


Figure 3. The tag responses in two subcarriers

3.3 Pseudo Code and an Example

The pseudo code of the proposed PRQT protocol is given in Fig. 4 to show the interactions of the reader and tags. Below are explanations of the pseudo code for the tag whose identification is denoted by *ID*.

On receiving request string S from the reader, the tag check whether or not ID prefix matches ($S\oplus MSB$), where ($S\oplus MSB$) stands for the result of executing the XOR operation on each bit of S with MSB, the most significant bit of ID. If so, the tag responds with R to the reader in subcarrier 0 (resp., 1) for the case of $R_1\oplus MSB = 0$ (resp., 1), where R=$Postfix$(ID, $|S|$+1) stands for the remainder or postfix of ID starting from the ($|S|$+1)th bit and R_1 stands for the first bit of R.

```
The pseudo code for the tag
//ID denotes the tag identification
//MSB denotes the most significant bit of ID
1.  On receiving request string S from the reader, the tag executes
2.    If (ID Prefix matches ($S\oplus MSB$))
3.      R=Postfix(ID, |S|+1)
4.      If ($R_1\oplus MSB = 0$) //$R_1$ stands for the first bit of R
5.        Respond with R in subcarrier 0
6.      Else
7.        Respond with R in subcarrier 1
8.      End If
9.  End If

The pseudo code for the reader
//SStack is a stack to keep request strings to interrogate tags
1.  If (Length(tag ID) =  odd)   //Initialize the SStack
2.    Push “01” and “00” into SStack
3.  Else
4.    Push “011”, “010”, “001”, and “000” into SStack
5.  End If
6.  While ($SStack\neq\varnothing$)
7.    Pop S from SStack
8.    Send S to all tags
9.    Wait to receive R from tags in subcarrier C, for C=0 and 1
      //R is the remainder or postfix of responding tag’s ID
      //$R_1$ is the 1st bit of R, and $R_2$ is the 2nd bit of R
10.   If (|R| =1) //only one bit is left to be identified
11.     If (collisions occur in subcarrier C)
12.       Identify a tag with ID=($S\oplus 0\oplus C$)+0 in subcarrier C
13.       Identify a tag with ID=($S\oplus 1\oplus C$)+1 in subcarrier C
14.     Else //only one tag responds in subcarrier C
15.       Identify a tag with ID=($S\oplus R\oplus C$)+R in subcarrier C
16.     End If
17.   Else  //two or more bits are left to be identified
18.     If (all R’s bits can be recognized) //no collision occurs in subcarrier C
19.       Identify a tag with ID=($S\oplus R_1\oplus C$)+R in subcarrier C
20.     Else
21.       If (R1 and R2 can be recognized in subcarrier C, but other bits collide)
22.         Push S +C+($R_1\oplus R_2\oplus C$) into SStack
23.       Else //Collisions occur in subcarrier C
24.         Push S+($0\oplus C$)+($1\oplus C$) and S+($0\oplus C$)+($0\oplus C$) into SStack
25.       End If
26.     End If
27.   End If
28. End While
```

Figure 4. The pseudo code of the PRQT protocol

Below are the explanations of the pseudo code for the reader:

- ■ Lines 1-5: The reader initializes *SStack*, which is a stack to keep request strings to interrogate tags. There are two cases for the initialization.
  - ● In one case, the tag ID is of an odd size and the strings “01” and “00” are pushed into *SStack* one by one.
  - ● In the other case, the tag ID is not of an odd size and the strings “011”, “010”, “001”, and “000” are pushed into *SStack* sequentially.

  The two initialization cases is due to the action of lines 10-16 for identifying tags whose tag ID remainder or postfix is only one bit, which will be explained later.
- ■ Lines 6-9: While *SStack* is not empty, the reader pops a string S from *SStack* and sends S to all tags for interrogating them. The reader then waits for a while to receive the tag ID postfix (or remainder) R from some tags.
- ■ Lines 10-16: These are for the case that only one ID bit is left to be identified (i.e., the length of tag ID remainder R is 1). The bit value may be one of three cases: 0, 1 or collision (it implies that 0 and 1 are sent and collide). For the three cases, the reader can respectively identify a tag with ID=$(S\oplus R_1\oplus C)+0$, identify a tag with ID=$(S\oplus R_1\oplus C)+1$, and identify two tags with IDs $(S\oplus R_1\oplus C)+0$ and ID=$(S\oplus R_1\oplus C)+1$ in subcarrier C, C=0 and 1.
- ■ Line 17-26: These are for the case that two or more bits are left to be identified.
- ■ Line 18-19: If no collision occurs in subcarrier C, C=0 or 1, then all bits of R can be recognized and the tag with ID=$(S\oplus R_1\oplus C)+R$ is identified.
- ■ Line 21-22: This is for the case that collisions occur but the first bit and the second bit can be recognized in subcarrier C, C=0 or 1.
  - ● For subcarrier 0, the responding tags are those whose ID prefix matches S with R_1R_2= “00” or R_1R_2 = “01”, and those whose ID prefix matches $\bar{S}$ with R_1R_2 = “10” or R_1R_2 = “11”.
  - ● For subcarrier 1, the responding tags are those whose ID prefix matches S with R_1R_2= “11” or R_1R_2 = “10”, and those whose ID prefix matches $\bar{S}$ with R_1R_2 = “00” or R_1R_2 = “01”.

  To sum up, the reader needs to append 2 bits to the request string S and push it into *SStack* for subcarrier C. For example, if only tags with R_1R_2= “00” respond in subcarrier 0, the reader pushes $S+C+(R_1\oplus R_2\oplus C)=S+$“00” into *SStack*. For another example, if only tags with R_1R_2= “10” respond in subcarrier 1, the reader pushes $S+C+(R_1\oplus R_2\oplus C)=S+$“10” into *SStack*.
- ■ Lines 23-24: This is for the case that collisions occur in subcarrier C, C=0 or 1. In such a case, the bit strings $S+(0\oplus \mathrm{C})+(1\oplus \mathrm{C})$ and $S+(0\oplus \mathrm{C})+(0\oplus \mathrm{C})$ are put into *SStack*.

An example of the PRQT protocol execution is depicted in Table 1 to show the tag interrogation steps and the associated interrogation tree for 8 tags with the same tag IDs given in the example for the QT protocol in Section II. Due to space limitation, only the first four steps are explained below. By those steps, it is observed that the number of iterations needed to identify all tags is decreased significantly.

In step 1, the reader sends the request string “00” and the tags with ID prefix being “00” will respond with the tag remainder *R*. The tag with ID=“0000100” will respond in subcarrier 0, since $R_1 \oplus MSB=0$. On the other hand, the tags with ID= “0010100”, “0011010” and “0011101” will respond in subcarrier 1, since $R_1 \oplus MSB=1$. Meanwhile, the tags with ID prefix being “11” will also respond. The tags with ID= “1100111” and “1101110” will respond in subcarrier 1, since $R_1 \oplus MSB=1$. Since no collision occurs in subcarrier 0, the tag with ID=“0000100” is identified successfully. However, collisions occur in subcarrier 1, so no tag can be identified and two more request strings “0011” (i.e., $S+(0\oplus \mathrm{C})+(1\oplus \mathrm{C})$) and “0010” (i.e., $S+(0\oplus \mathrm{C})+(0\oplus \mathrm{C})$) are sequentially pushed into *SStack*.

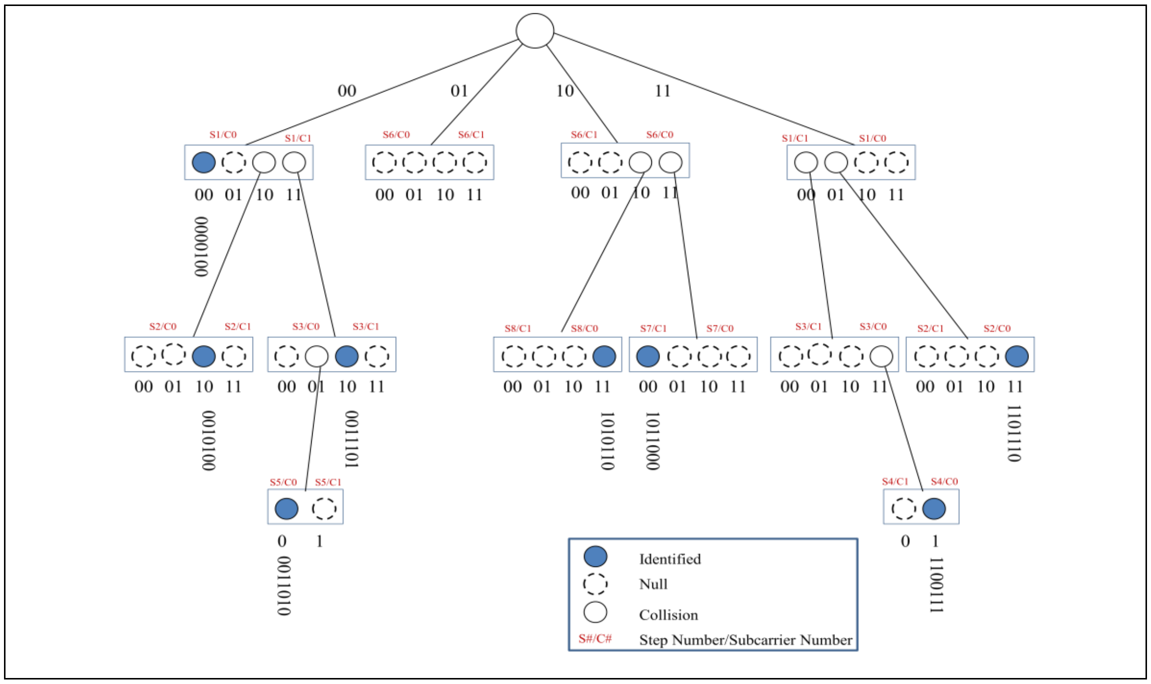
In step 2, the request string *S*=“0010” is popped from *SStack* and sent to all tags. The tag with ID =“0010100” responds in subcarrier 1, since its ID prefix matches S and $R_1 \oplus MSB=1$. And the tag with ID =“1101110” responds in subcarrier 0, since its ID prefix matches $\bar{S}$ and $R_1 \oplus MSB=0$. No collision occurs while the two tags respond, so they can be identified successfully.

In step 3, the request string *S*=“0011” is popped from *SStack* and sent to all tags. Only the tag with ID=“0011101” responds in subcarrier 1, so it is identified successfully. However, tags with ID=“0011010” or ID=“1100111” respond in subcarrier 0, so they cannot be identified successfully and two more request strings “001101” (i.e., $S+(0\oplus \mathrm{C})+(1\oplus \mathrm{C})$) and “001100” (i.e., $S+(0\oplus \mathrm{C})+(0\oplus \mathrm{C})$) are sequentially pushed into *SStack*.

In step 4, the request string *S*=“001100” is popped from *SStack* and sent to all tags. Only the tag with ID=“1100111” responds in subcarrier 0, so it is identified successful. Note that $|R|=1$, so even if collisions occur in a subcarrier, the responding tags can also be identified successfully.

Table 1. An example of PRQT protocol execution

Step	Request string *S*	ID	Response		Result
		0000100 0010100 0011010 0011101 1010110 1011000 1100111 1101110			
1	00		subcarrier 0	00100	0000100 identified
			subcarrier 1	10100 11010 11101 00111 01110	Collision
2	0010		subcarrier 0	110	1101110 identified
			subcarrier 1	100	0010100 identified
3	0011		subcarrier 0	010 111	Collision
			subcarrier 1	101	0011101 identified
4	001100		subcarrier 0	1	1100111 identified
			subcarrier 1		Null
5	001101		subcarrier 0	0	0011010 identified
			subcarrier 1		Null
6	01		subcarrier 0	10110 11000	Collision
			subcarrier 1		Null
7	0100		subcarrier 0		Null
			subcarrier 1	000	1011000 identified
8	0101		subcarrier 0	110	1010110 identified
			subcarrier 1		Null
Interrogation Tree					



4. Analysis

In this section, the PRQT protocol is analyzed in terms of the number of iterations needed to interrogate all the tags. It is assumed that the number *n* of tags is far smaller than the size $N=2^{(\text{length of tag ID})}$ of the tag ID space. The analysis is based on the fact that the identification tree of the PRQT protocol is a 4-ary tree, which can be observed by the example shown in Table 1. Two tag ID distribution cases are considered in the following subsections: the uniform tag ID distribution and the consecutive tag ID distribution.

4.1 Uniform tag ID distribution

To simplify the analysis, the number *n* of tags is assumed to be a power of 4. Please refer to Fig. 5, the total iterations needed to identify all tags are those included in the gray triangular area and white trapezoid area of the tag ID space. Below, the number of iterations in the gray triangular area of the tag ID space is first analyzed. For a 4-ary tree, it can be assumed that *n*/4 tags are in a level-1 node, *n*/16 tags are in a level-2 node, …, and 1 tag is in a level-$\log_4 n$ node for the uniform tag distribution case. When the reader sends out the request string *S*, the tags with ID prefixes matching one of the tag ID prefixes $S00$, $S01$, $S10$, $S11$, $\bar{S}00$, $\bar{S}01$, $\bar{S}10$, $\bar{S}11$ will respond, leading to collisions. The reader then needs to send longer request strings $S00$, $S01$, $S10$, $S11$ to identify tags. Therefore, the total number of iterations in the gray triangular area of the tag ID space is as follows.

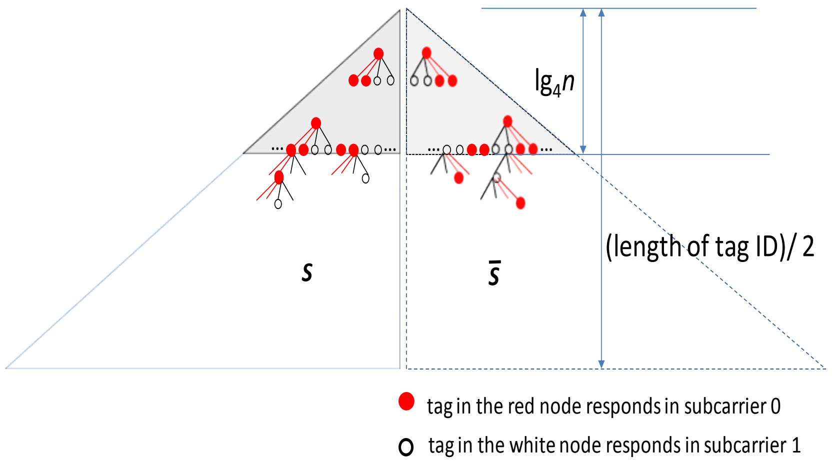
$$I_{gray\_area} = \frac{1}{2}\left(\sum_{i=1}^{\lg_4 n-1} 4^i\right) = \frac{2}{3}\ \left(4^{\lg_4 n-1} - 1\right) \qquad (1)$$

The number of iterations in the white trapezoid area of the tag ID space is now analyzed. When the length of the request string S sent from the reader is $\log_4 n$, only two tags will have prefixes matching S or $\bar{S}$. The two tags are either a tag with ID prefix in $\{S00, S01, S10, S11\}$ or a tag with ID prefix in $\{\bar{S}11, \bar{S}10, \bar{S}01, \bar{S}00\}$. As described in Table 2, in all the possible pairs of these two tag IDs, half of them lead to successful tag identification, and others, collisions. This means there is only 1/2 probability that the reader needs to send longer request strings with the length of $\lg_4 n + 1$ later. The identification procedure will repeat until all tags are identified successfully. The total number of iterations associated with the white trapezoid area of the tag ID space is thus as follows.

$$I_{white\_area} = \frac{1}{2}\left(\sum_{k=1}^{\lg_4 N - \lg_4 n - 1} \frac{n}{2^k}\right) = \frac{n}{2}\left(1 - 2^{-\lg_4 \frac{N}{4n}}\right) \tag{2}$$

Hence, the total number of iterations I_n to identify all tags n in the interrogation zone is as follows.

$$I_n = I_{gray\_area} + I_{white\_area} = \frac{2}{3}\ \left(4^{\lg_4 n - 1} - 1\right) + \frac{n}{2}\left(1 - 2^{-\lg_4 \frac{N}{4n}}\right) \tag{3}$$



lg$_4n$
(length of tag ID)/ 2
S
$\bar{S}$
● tag in the red node responds in subcarrier 0
○ tag in the white node responds in subcarrier 1

Figure 5. The identification tree with the uniform tag ID distribution

Table 2. Combinations of tag ID prefixes for the request string of length $\lg_4 n + 1$

Combination no	Tag ID prefix	Response in which subcarrier	Tag ID prefix	Response in which subcarrier	Result
1	$S00$	0	$\bar{S}11$	0	Collision
2	$S00$	0	$\bar{S}10$	0	Collision
3	$S00$	0	$\bar{S}01$	1	Identified
4	$S00$	0	$\bar{S}00$	1	Identified
5	$S01$	0	$\bar{S}11$	0	Collision
6	S01	0	$\bar{\text{S}}10$	0	Collision
7	S01	0	$\bar{\text{S}}01$	1	Identified
8	$S01$	0	$\bar{S}00$	1	Identified
9	$S10$	1	$\bar{S}11$	0	Identified
10	$S10$	1	$\bar{S}10$	0	Identified
11	$S10$	1	$\bar{S}01$	1	Collision
12	$S10$	1	$\bar{S}00$	1	Collision
13	$S11$	1	$\bar{S}11$	0	Identified
14	$S11$	1	$\bar{S}10$	0	Identified
15	$S11$	1	$\bar{S}01$	1	Collision
16	$S11$	1	$\bar{S}00$	1	Collision

4.2 Consecutive tag ID distribution

To simplify the analysis, it is assumed that the length of tag ID is odd and the number of tags *n* is a power of 4. Two extreme cases depicted in Fig. 6(a) and (b) are discussed. In Fig. 6(a), the distribution of tag IDs is consecutive and symmetric to the median of the tag ID space. At the first iteration when *S* = “00” is issued, no response is received in both subcarriers because no tag has the ID prefix matching $S00$, $S01$, $S10$, $S11$, $\bar{S}00$, $\bar{S}01$, $\bar{S}10$, or $\bar{S}11$. Afterwards, *S* = “01” is issued and no tag has the ID prefix matching $S00$, $S01$, $S10$, $\bar{S}01$, $\bar{S}10$, or $\bar{S}11$ except $S11$ and $\bar{S}00$. The tags with the ID prefixes matching $S11$ or $\bar{S}00$ will simultaneously respond with the ID postfix starting from the 3rd bit in subcarrier 1. Because collisions occur in subcarrier 1, the reader then needs to send longer request strings 0110 or 0111 later. When the reader sends request string *S* = “0110”, no tag responds. However, while the reader sends *S* = “0111”, tags with the

ID prefix matching “0111” or “1000” respond in subcarrier 1. The identification procedure will repeat until the length of S becomes $\lg_4 N - \lg_4 n$ (i.e., the top of the gray triangular area in the tag ID space in Fig. 6(a) is reached). The number I_{white_area} of iterations in the white area of the tag ID space in Fig. 6(a) is thus as follows.

$$I_{white\_area} = 2\left(\lg_4 N - \lg_4 n - 1\right) = \lg_4 \left(\frac{N}{4n}\right)^2 \tag{4}$$

The number of iterations in the gray area of the tag ID space in Fig. 6(a) is now analyzed. When a request string S with the length of $\lg_2 N - \lg_2 n$ is issued, tags with ID prefixes matching $S00$, $S01$, $\bar{S}10$, or $\bar{S}11$ will respond in subcarrier 0, while tags with ID prefixes matching $S10$, $S11$, $\bar{S}00$, or $\bar{S}01$ will respond in subcarrier 1. The reader then needs to send longer request strings $S00$, $S01$, $S10$, and $S11$ later. The number $I_{\mathrm{gray\_area}}$ of iterations in the gray area of the tag ID space will be

$$I_{gray\_area} = \frac{1}{2}\left(\sum_{i=1}^{\lg_4 n-1} 4^i\right) = \frac{2}{3} \left(4^{\lg_4 n-1}\right) \tag{5}$$

The total number of iterations I_n is thus as follows.

$$I_n = I_{white\_area} + I_{gray\_area} = \lg_4 \left(\frac{N}{4n}\right)^2 + \frac{2}{3} \left(4^{\lg_4 n-1}\right) \tag{6}$$

In Fig. 6(b), the tag IDs are consecutive but are all in half part of the tag ID space. The number of iterations in the white area of the tag ID space in Fig. 6(b) is first analyzed. When the request string S=“00” is issued, all tags will respond simultaneously in subcarrier 0 but no tag responds in subcarrier 1. When S=“01” is issued, no tag responds in either subcarriers. The reader thus needs to send longer request string “0000” later. The identification procedure will repeat until the length of S is $\log_4 N - \log_4 n$ (i.e., the gray area of the tag ID space in Fig. 6(b) is reached). The number I_{white_area} of iterations in the white area of the tag ID space in Fig. 6(b) is thus as follows.

$$I_{white\_area} = 2 + \lg_4 N - \lg_4 n = \lg_4 \left(\frac{16N}{n}\right) \tag{7}$$

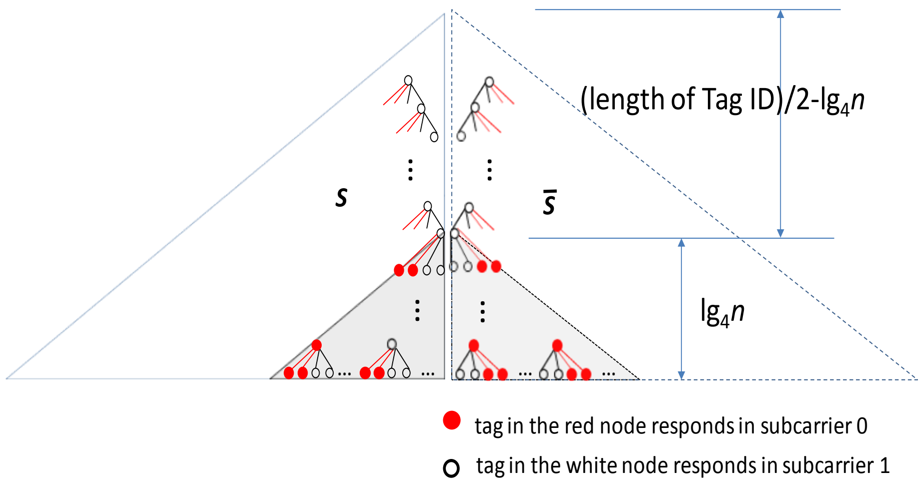
Now, the number $I_{\mathrm{gray\_area}}$ of iterations in the gray area of the tag ID space in Fig. 6(b) is analyzed. When S with the length of $\lg_4 N - \lg_4 n$ is issued (i.e., the top of the gray area in the tag ID space is reached), tags with the ID prefix $S00$ or $S01$ will respond in subcarrier 0, while $S10$ or $S11$ will respond in subcarrier 1.

The reader then needs to send longer request strings $S00$, $S01$, $S10$, and $S11$ later. The number $I_{\text{gray\_area}}$ of iterations is thus as follows.

$$I_{gray\_area} = \frac{1}{2}\left(\sum_{i=1}^{\lg_4 n-1} 4^i\right) = \frac{2}{3}\left(4^{\lg_4 n-1} - 1\right) \tag{8}$$

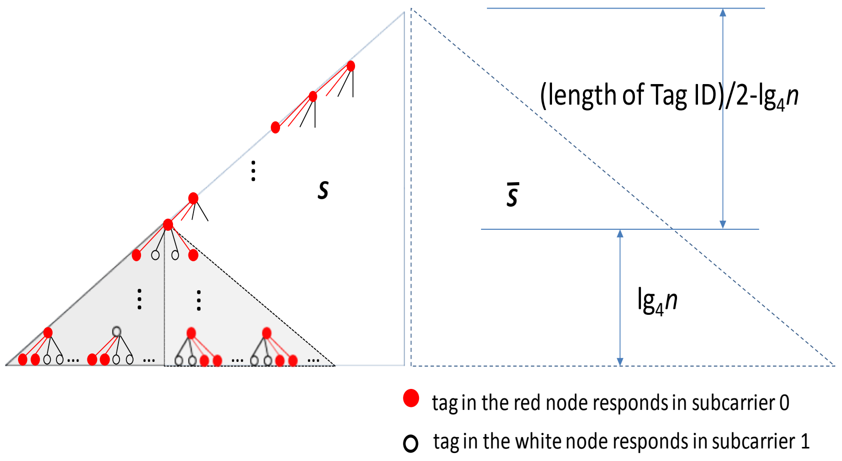
The total number I_n of iterations is as follows.

$$I_n = I_{white_{area}} + I_{gray_{area}} = \lg_4\left(\frac{16N}{n}\right) + \frac{2}{3}\left(4^{\lg_4 n-1} - 1\right) \tag{9}$$



(length of Tag ID)/2-$\lg_4 n$
s
$\bar{s}$
$\lg_4 n$
● tag in the red node responds in subcarrier 0
o tag in the white node responds in subcarrier 1

(a) Tag IDs are consecutive and symmetric to the median of the tag ID space



(length of Tag ID)/2-$\lg_4 n$
s
$\bar{s}$
$\lg_4 n$
● tag in the red node responds in subcarrier 0
o tag in the white node responds in subcarrier 1

(b) Tag IDs are consecutive but all in a half part of the tag ID space

Figure 6. The illustration of the identification trees with the consecutive tag ID distribution

5. SIMULATION

In this section, the impacts of the ID length, the ID distribution and the number of tags on the PRQT protocol performance, which is measured in terms of the number of identification iterations and transmission bits, is investigated by simulations. The simulation results are also compared with those of related protocols. Note that simulation experiments are performed 1000 times for each case of settings.

5.1 The impact of the number of tags under the uniform ID distribution

This subsection shows the simulation results about the number of iterations needed to identify 1, 50, 100, 150,…, 1000 tags within the interrogation zone. It is assumed that tag IDs are uniformly distributed and the length of tag ID is 10, 16, 32, 64, or 128. The simulation results are shown in Fig. 7, by which it can be observed that when tag IDs are distributed sparsely over the ID space (e.g., the ratio of the number of tags to the tag ID space is less than 0.25), the number of iterations increases linearly with the number of tags and is not affected by the length of the tag ID (i.e. the size of the tag ID space). On the contrary, when tag IDs are distributed densely over the tag ID space, the number of iterations is obviously independent of the number of tags.

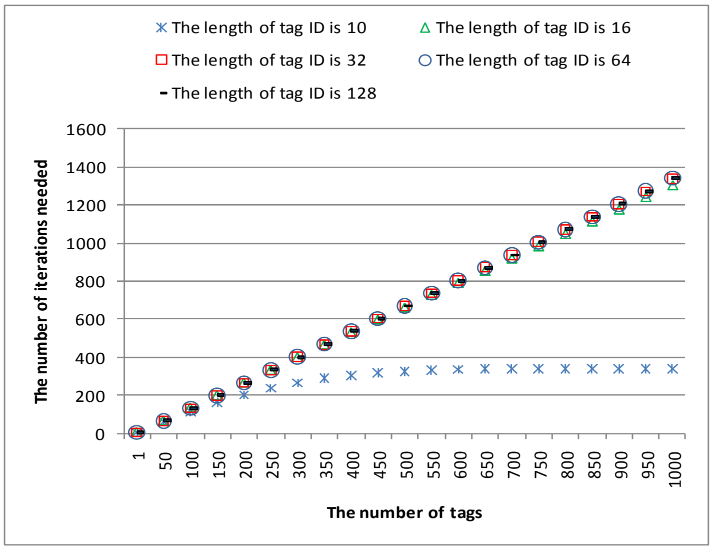


Figure 7. The relationship between the number of iterations and the number of tags under the uniform tag ID distribution

5.2 The impact of number of tags under consecutive ID distribution

This subsection describes the second set of simulations under the same assumption as adopted in Section 5.1, except that the tag ID distribution is assumed to be consecutive. From Fig. 8, it is observed that when tag IDs are sparsely distributed over the tag ID space (e.g., the ratio of the number of tags to the size of the tag ID space is less than 0.5), the number of iterations is about 50% less than that of the uniform distribution case. When the tags are dense relatively to the ID space (e.g., the ratio of the number of tags to the size of the tag ID space is larger than 0.5), the number of iterations needed is obviously not affected by the number of tags and is nearly the same as that of the uniform tag ID distribution case.

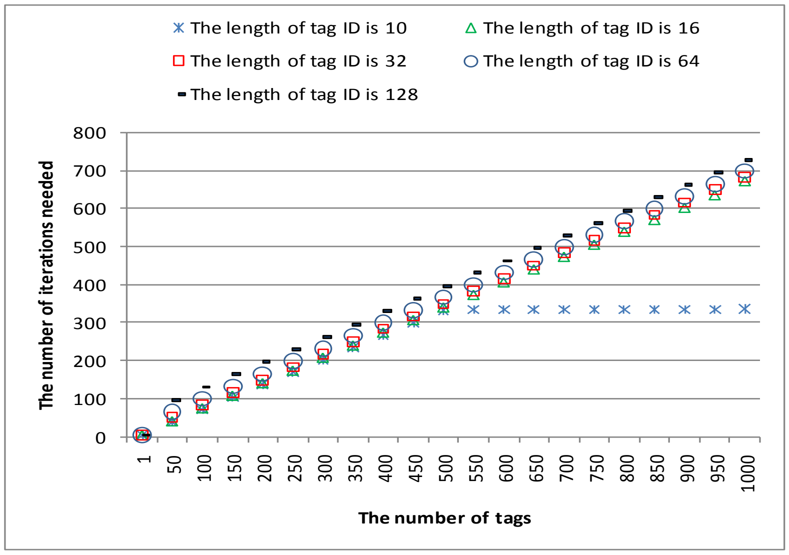


Figure 8. The relationship between the number of iterations and the number of tags under the consecutive tag ID distribution

5.3 The impact of tag ID length under uniform ID distribution

This subsection describes the set of simulations addressing the relations between the tag ID length and the number of iterations needed for tag identification. The tag ID length ranges from 11 to 128 for four cases of 500, 1000, 1500 and 2000 tags. The simulation results are plotted in Fig. 9, from which it can be observed that when the ratio of the number of tags to the size of the tag ID space is under 0.2, the number of iterations is independent of the tag ID length and is about 1.34 times the number of tags.

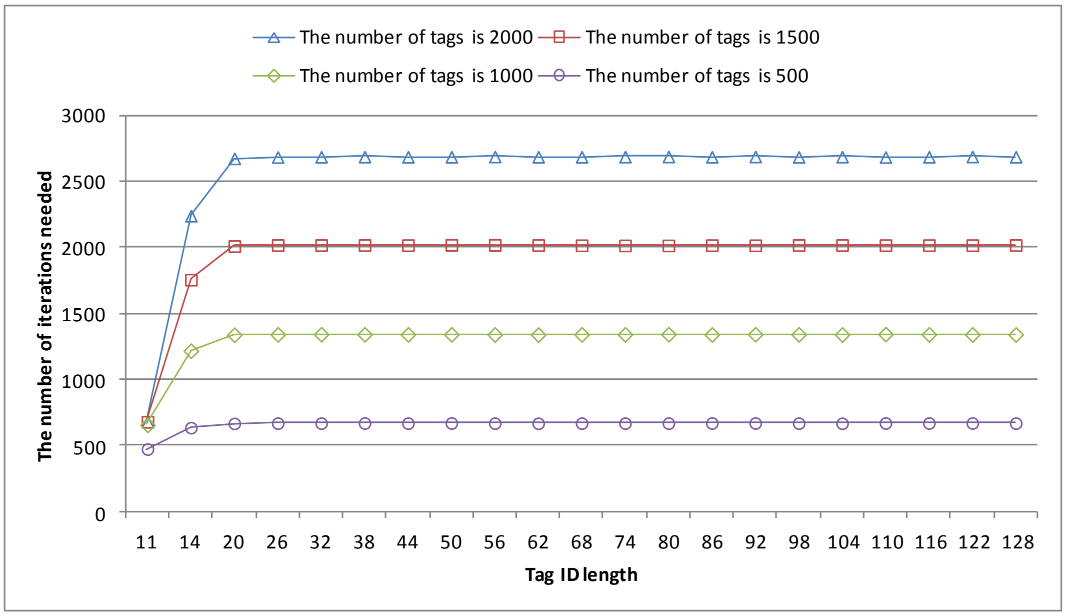


Figure 9. The relationship between the number of iterations and the tag ID length under uniform tag ID distribution

5.4 The impact of tag ID length under the consecutive ID distribution

This subsection describes the set of simulations under the same assumption as adopted in Section 5.3, except for the tag ID distribution being consecutive. The simulation results are plotted in Fig. 10, from which it can be observed that when the ratio of the number of tags to the tag ID space is under 0.2, the number of iterations needed is affected slightly by the tag ID length. The number of iterations only increases by 1 when the length of the tag ID is 2 bits longer.

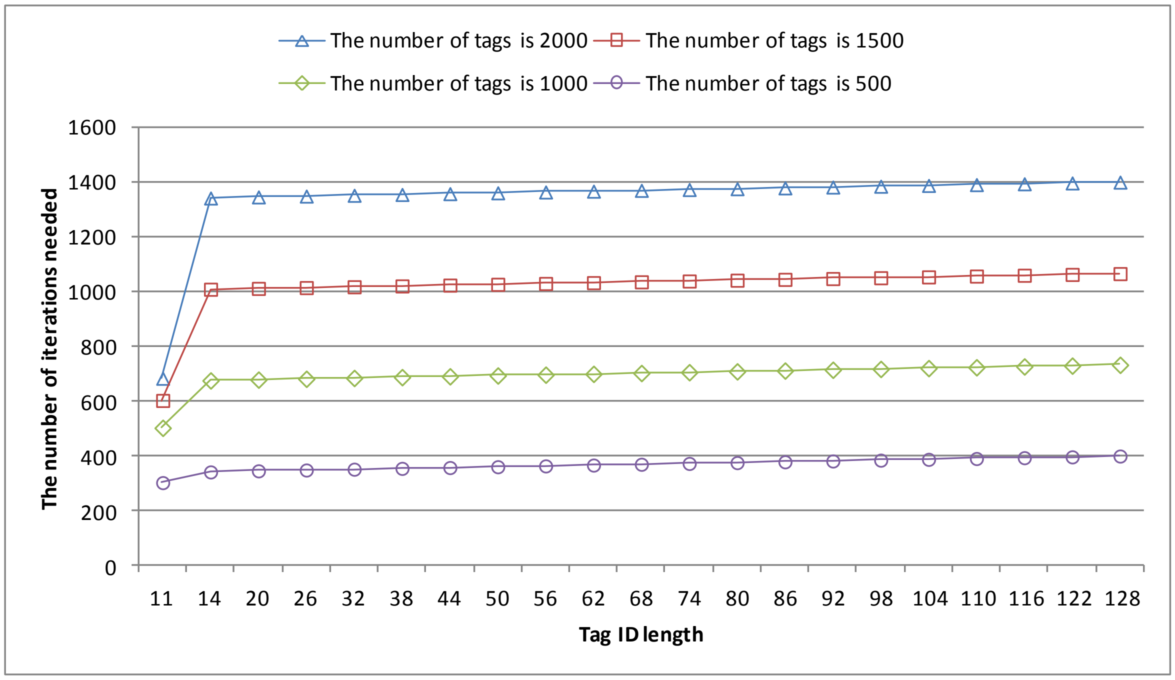


Figure10. The relationship between the number of iterations and tag ID length under the consecutive tag ID distribution

5.5 Comparisons of the number of identification iterations

This subsection shows the simulation result comparisons of the PRQT, QT, 4QT (4-ary query tree), BSQTA (Bi-slotted query tree algorithm), and CTTA protocols in terms of the number of iterations needed to identify all tags in the interrogation zone. The AQS protocol is not included in the comparisons, since it tries to improve efficiency on the basis of information gathered from the previous round of tag identification, while the PRQT protocol and other protocols try to improve efficiency within a specific round alone. The simulations are performed for 100, 150,…, 1000 tags in the interrogation zone under the assumption that tag ID is 64 bit long and are uniformly distributed. The simulation results are plotted in Fig. 11, by which it can be observed that the PRQT protocol requires less iterations than others. For example, the PRQT protocol needs around 47% iterations of the query tree and the 4-ary query tree protocols, and needs around 94% iterations of the bi-slotted query tree algorithm.

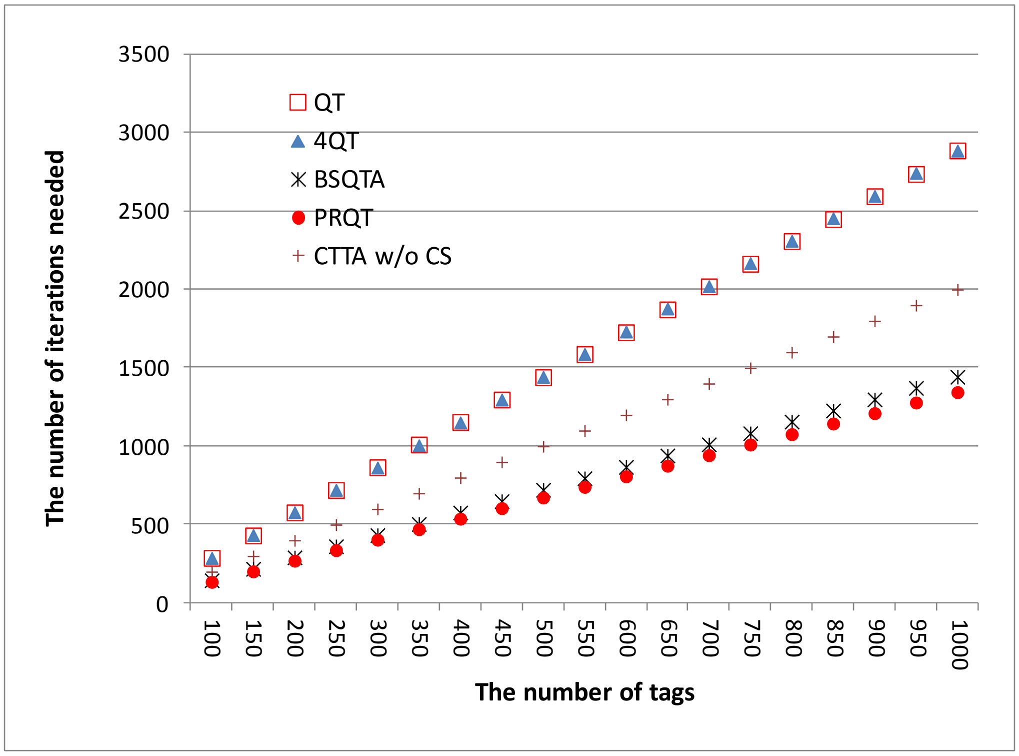


Figure 11. The numbers of iterations for different protocols

5.6 Comparisons of the number of transmission bits

This subsection shows the comparisons of the PRQT, QT, 4QT (4-ary query tree), BSQTA (Bi-slotted query tree algorithm), and CTTA protocols in terms of the number of transmission bits needed to identify all tags in the interrogation zone. As mentioned in relative work, it is impossible for low cost tags, such as those conforming to the EPCglobal C1 G2 standard, to transmit the tag ID and receive signals simultaneously in the CCTA protocol. As such, the simulations for the CTTA protocol without collision stop signaling (CTTA w/o CS) are also conducted. The simulations are performed for 100, 150, …, 1000 tags under the assumption that the tag ID is 64 bit long and uniformly distributed. The simulation results are plotted in Fig. 12, by which it can be observed that the PRQT protocol requires less transmission bits than others except CTTA. The PRQT protocol needs around 43%, 40% and 88% of transmission bits of the bi-slotted query tree algorithm, 4-ary query tree, and CTTA protocols without collision stop signaling, respectively. This is because the PRQT protocol allows tags to respond with IDs cleverly in two subcarriers in parallel within an iteration.

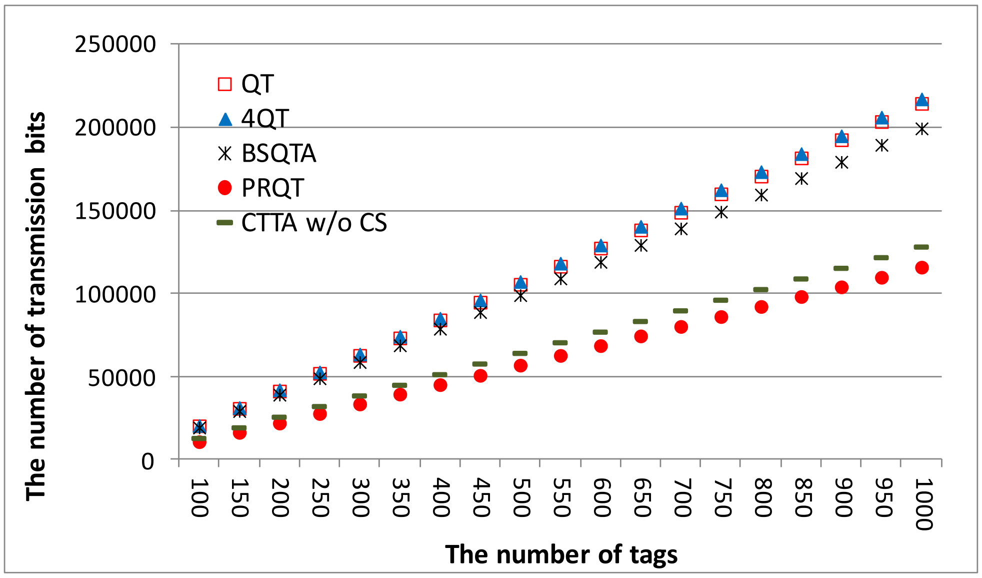


Figure 12. The number of transmission bits needed for different protocols

6. Conclusion

In this paper, a novel tree-based anti-collision protocol, called the parallel response query tree (PRQT) protocol, is proposed on the basis of two schemes: parallel prefix matching and parallel subcarrier responding. The first scheme allows tags whose ID prefixes matching the request string *S* or the complementary of *S* to respond with their ID postfixes at the same time. In order for the reader to distinguish the responses from these two groups of tags, the second scheme is applied to make tags respond in two subcarriers in parallel. Because tags need to parallelly match and report tag IDs, the circuit design in tags is more complex, leading to higher tag costs. Furthermore, the reader needs the ability to discriminate between signals transmitted in two subcarriers concurrently, causing more sophisticated and expensive readers. However, the proposed PRQT protocol indeed offers good performance. The PRQT protocol is analyzed in terms of the number of iterations needed to identify all tags. It and related protocols, such as the QT, 4QT, BSQTA, and CTTA protocols, are also simulated and compared. The simulation results show that the PRQT protocol is better than related ones in terms of the number of identification iterations and the number of transmission bits needed to identify all tags in the interrogation zone.

REFERENCES

[1] K. Finkenzeller, RFID handbook: Radio frequency identification fundamentals and applications, first ed., John Wiley & Sons 2000.

[2] N. Abramson, The ALOHA system - another alternative for computer communications, Proc. of AFIPS Spring Joint Computer Conf., Vol. 37, 1970, pp. 281-285.

[3] Leian Liu, Shengli Lai, ALOHA-Based Anti-Collision Algorithms Used in RFID System, Proc. of International Conf. on Wireless Communications, Networking and Mobile Computing (WiCOM 2006), Sep. 2006, pp.1-4.

[4] H. Choi, J. R. Cha, and J. H. Kim, Fast wireless anti-collision algorithm in ubiquitous ID system, Proc. of IEEE VTC '04, Sep. 2004.

[5] Ming-Kuei Yeh, Jehn-Ruey Jiang, Parallel Splitting for RFID tag anti-collision, International Journal of Ad Hoc and Ubiquitous Computing, Vol. 8, No. 4, 2011, pp. 249-260.

[6] ISO/IEC, Information technology automatic identification and data capture techniques – radio frequency identification for item management air interface - part 6: parameters for air interface communications at 860-960 MHz, Final Draft International Standard ISO 18000-6, Nov. 2003.

[7] C. Law, K. Lee, and K. Siu, Efficient Memoryless Protocol for Tag Identification, Proc. of the Fourth Int'l Workshop Discrete Algorithms and Methods for Mobile Computing and Comm., Aug. 2000

[8] Xinqing Yan, Guiliang Zhu, EBQP: Enhanced Binary Query Protocol for RFID tag collision resolution with progressive population estimation, International Journal of Ad Hoc and Ubiquitous Computing, Vol. 8, 2011, pp.78-84

[9] Jiho Ryu, Hojin Lee, Yongho Seok, Taekyoung Kwon, and Yanghee Choi, A Hybrid Query Tree Protocol for Tag Collision Arbitration in RFID systems, Prof. of IEEE International Conference on Communications, June 2007, pp.5981-5986.

[10] Draft protocol specification for a 900 MHz Class 0 Radio Frequency Identification Tag, MIT Auto-ID Center, Feb 2003.

[11] J.S. Cho, J.D. Shin and S.K. Kimg, RFID Tag Anti-Collision Protocol: Query Tree with Reversed IDs, Proc. of the 10th International Conference on Advanced Communication Technology, February 2008, pp. 225-230.

[12] Yun Tian, Gongliang Chen, Jianhua Li,An, Effective Temporary ID Based Query Tree Algorithm for RFID Tag Anti-collision, Internet of Things Communications in Computer and Information Science, Vol. 312, 2012, pp. 242-247.

[13] Ji Hwan Choi, Dongwook Lee, Hyuckjae Lee, Bi-slotted tree based anti-collision protocols for fast tag identification in RFID systems, IEEE Communications Letters, Vol. 10, Issue 12, 2006, pp. 861-863.

[14] Jihoon Myung, Wonjun Lee, Adaptive splitting protocols for RFID tag collision arbitration, Proc. of MobiHoc 2006, 2006, pp. 202-213.

[15] F. Zhou, D. Jin, C. Huang, and M. Hao, Optimize the Power Consumption of Passive Electronic Tags for Anti-collision Schemes, Proc. of the 5th Int'l Conf. on ASIC, Vol. 2, Oct. 21-24, 2003, pp.1213-1217.

[16] P.Mathys and P.Flajolet, Q-ary collision resolution algorithms in random-access systems with free or blocked channel access, IEEE Trans. Inform. Theory, Vol. 31, No. 4, March 1985, pp. 217-243.

[17] Jehn-Ruey Jiang and Ming-Kuei Yeh, Anti-collision protocols for the RFID system, in: Yan Zhang et al. (Eds.), RFID and Sensor Networks: Architectures, Protocols, Security and Integrations, Auerbach Publications, Taylor&Francis Group, USA, 2009.

[18] H. Gou, H. C. Jeong, and Y. Yoo, A bit collision detection based query tree protocol for anti-collision in RFID system, Prof. of IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications, Oct. 2010, pp. 421-428.

[19] Ji Hwan Choi, Dongwook Lee, Hyoungsuk Jeon, Jongsub Cha, and Hyuckjae Lee, Enhanced Binary Search with Time-Divided Responses for Efficient RFID Tag Anti-Collision, Proc. of the IEEE International Conference on Communications, June 2007, pp. 3853-3858.

[20] Chiu-Kuo Liang, Hsin-Mo Lin, Using Dynamic Slots Collision Tracking Tree Technique Towards an Efficient Tag Anti-Collision Algorithm in RFID Systems, Proc. of the 9th International Conference on Ubiquitous Intelligence & Computing/Autonomic & Trusted Computing (UIC/ATC), 2012, pp. 272-277.

[21] Xiaolin Jia, Quanyuan Feng, Lishan Yu, Stability Analysis of an Efficient Anti-Collision Protocol for RFID Tag Identification, IEEE Transactions on Communications, Vol. 60, No. 8, 2012, pp.2285-2294.

[22] EPCglobal Ratified Standard, EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz, EPCglobal Standards, Oct. 2007.

[23] 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

[24] Pete Sorrells, Passive RFID Basics, Microchip Technology Inc., 1998.