A Novel Query Tree Protocol Based on Partial Responses for RFID Tag Anti-Collision

Ming-Kuei Yeh
Department of Information Management
National Taipei College of Business
Taipei, Taiwan
jamesyeh@webmail.ntcb.edu.tw

Jehn-Ruey Jiang
Department of Computer Science and Information Engineering
National Central University
Jhongli, Taiwan
jrjiang@csie.ncu.edu.tw

***Abstract*—In the RFID system, when two or more tags respond their IDs to the reader simultaneously, wireless signal collision occurs and no tag can be identified successfully by the reader. How to reduce such collisions in order to speed up the identification performance is thus important. There are many anti-collision protocols proposed to solve the tag collision problem. They can be categorized into two classes: ALOHA-based and tree-based protocols. The query tree (QT) protocol is a famous tree-based protocol having many advantages. It is stateless and uses no on-tag memory to keep protocol states; it is a plain protocol and uses no special techniques, such as bit-tracking, ID-revising, and re-identification. In this paper, we propose a stateless and plain tree based anti-collision protocol, called PRQT, by using tag ID partial responses to speed up tag identification. We also conduct simulation experiments for PRQT and compare it with QT in terms of the number of iterations to identify tags. As we will show, the PRQT protocol uses less numbers of iterations to identify tags than the QT protocol.**

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

I. INTRODUCTION

The front end of an RFID (Radio Frequency IDentification) system is composed of readers and tags. When a tag and a reader are close enough, they can communicate with each other and the tag is said in the *interrogation zone* of the reader. The reader can initiate an *interrogation procedure* (or *identification procedure*) to send interrogation request to ask tags to respond with their IDs. When multiple tags respond to the reader request simultaneously, signal collisions occur and no tag can be identified by the reader successful. How to reduce tag signal collisions to speed up the interrogation procedure is thus important.

Several anti-collision protocols are proposed to reduce tag signal collisions. They can be categorized into two classes: ALOHA-based protocols and tree-based protocols. The principle of ALOHA-based protocols [1, 2] is that each tag, on receiving the reader's interrogation request, independently chooses a random back-off time and responds its tag ID to the reader at that time. If no collision occurs during a tag's ID response, its ID can be identified successfully and acknowledged by the reader. A tag with acknowledged ID will stop responding to the reader. On the other hand, an unacknowledged tag will repeatedly select a random back-off time and respond with its ID until it is identified and acknowledged by the reader. In tree-based protocols [3, 4], a group of tags is split into subgroups according to the reader's request and/or the condition of tag signal collisions. The splitting will continue until there is only one tag in a subgroup to be identified successfully.

This paper focuses on the well-known query tree (QT) protocol [4], which is a stateless and plain protocol. It is stateless since it uses no on-tag memory to keep protocol states; it is plain since it uses no special techniques, such as bit-tracking, ID-revising, and re-identification. Based on the QT protocol, we propose a stateless and plain tree based anti-collision protocol, called PRQT, by using tag ID partial responses to speed up tag identification. We also conduct simulation experiments for PRQT and compare it with QT in terms of the number of iterations to identify tags. As we will show, PRQT outperforms QT.

The rest of this paper is organized as follows. The QT protocol is described in Section II, and the PRQT protocol is proposed in Section III. PRQT is simulated and compared with related ones in Section IV. And finally, conclusions are drawn in Section V.

II. THE QT PROTOCOL AND ITS VARIANTS

The QT protocol [4] is a tree-based RFID anti-collision protocol. It is stateless and uses no on-tag memory to keep protocol states; it is a plain protocol and uses no special techniques, such as bit-tracking, ID-revising, and re-identification for tag identification procedure. To initiate the identification procedure, the reader first pushes request strings "1" and "0" into the stack, then pops up a request string *S* from the stack and broadcasts it to tags to start a *round* or an *iteration* of the identification procedure. The tag with the ID prefix matching the request string *S*, responds to the reader with its ID remainder. If only one tag responds, it will be identified successfully; otherwise, tag signals collide and no tag can be identified. In the case of multiple tag responses, the reader pushes two longer request strings, "*S*1" and "*S*0", into the stack so that the tags encountering signal collisions can be split into two subgroups in the following rounds. The identification procedure continues to identify all tags until the stack is empty.

In Table I, we give an example of the QT protocol to explain the details of the identification procedure [5]. Suppose there are 8 tags in the interrogation zone of the reader, whose IDs are uniquely 0001010, 0010101, 0100100, 0110100, 1010110, 1011000, 1100111, and 1101010. Below, we describe the first 10 rounds of the identification procedure, where a round contains the following actions: (1) the reader pops up a string and sends it to all tags, (2) zero, one or more tags respond, and (3) the reader processes tag responses.

Initially: The reader pushes request strings "1" and "0" into the stack.

Round 1: The reader pops up and sends the request string *S*="0" to tags. The tags with IDs 0001010, 0010101, 0100100 and 0110100 send the last 6 bits of the IDs to the reader since the prefix "0" of the IDs matches with *S*. In such a case, tag signals collide and no tag can be identified successfully. The reader pushes request strings "01" and "00" into the stack.

Round 2: The reader pops up and sends the request string *S*="00" to tags. The tags with IDs 0001010 and 0010101 send the last 5 bits of the IDs to the reader since the prefix "00" of the IDs matches "00". Signal collision occurs again, and no tag can be identified successfully. The reader pushes request strings "001" and "000" into the stack.

Round 3: The reader pops up and sends the request string *S*="000" to tags. Only the tag with ID 0001010 sends the last 4 bits of the ID to the reader since the prefix "000" of the ID matches with *S*. Since only one tag responds, it is identified successfully.

Round 4: Again, the reader pops up and sends the request string *S*="001" to tags. Only one tag with ID 0010101 responds and it is thus identified successfully.

Round 5: The next request string popped up is "01". The tags with IDs 0100100 and 0110100 send the last 5 bits of the IDs to the reader since the prefix "01" of the IDs matches *S*. Tag signals collide and no tag can be identified successfully. The reader pushes request strings "011" and "010" into the stack.

Round 6: The reader pops up and sends the request string *S*="010" to tags. The tag with ID 0100100 responds, since the ID prefix "010" matches with *S*. It is the only tag that responds and is thus identified successfully.

Round 7: The request string *S*="011" is popped up and sent to tags. Only the tag with ID 0110100 responds; it is identified successfully.

Round 8: The request string *S*="1" is popped up and sent to tags. At this time, tags with IDs 1010110, 1011000, 1100111 and 1101010 respond simultaneously. Tag signals collide and no tag can be identified successfully. The reader then pushes request strings "11" and "10" into the stack.

Round 9: The request string *S*="10" is popped up and broadcasted. Tags with IDs 1010110 and 1011000 respond and signal collide. The reader pushes request strings "101" and "100" into the stack.

Round 10: Again, the reader pops up and send the request string *S*="100" to tags. Because no tag is with the ID prefix matching *S*, no tag respond.

The identification procedure proceeds round by round until the stack is empty. When the stack is empty, all the tags are identified successfully and the identification procedure of the QT protocol is finished.

TABLE I. AN EXAMPLE OF THE QT PROTOCOL EXECUTION FOR TAGS WITH IDS 0001010, 0010101, 0100100, 0110100, 1010110, 1011000, 1100111 AND 1101010.

Round/ Iteration	Request string *S*	Stack	Response		Result
			Responding tag	*Responding bits*	
–	–	0 1	–	–	–
1	0	00 01 1	0001010 0010101 0100100 0110100	001010 010101 100100 110100	Collision
2	00	000 001 01 1	0001010 0010101	01010 10101	Collision
3	000	001 01 1	0001010	1010	0001010 identified
4	001	01 1	0010101	0101	0010101 identified
5	01	010 011 1	0100100 0110100	00100 10100	Collision
6	010	011 1	0100100	0100	0100100 identified
7	011	1	0110100	0100	0110100 identified
8	1	10 11	1010110 1011000 1100111 1101010	010110 011000 100111 101010	Collision
9	10	100 101 11	1010110 1011000	10110 11000	Collision
10	100	101 11	–	–	Null
11	101	1010 1011 11	1010110 1011000	0110 1000	Collision
12	1010	1011 11	1010110	110	1010110 identified
13	1011	11	1011000	000	1011000 identified
14	11	110 111	1100111 1101010	00111 01010	Collision
15	110	1100 1101 111	1100111 1101010	0111 1010	Collision
16	1100	1101 111	1100111	111	1100111 identified
17	1101	111	1101010	010	1101010 identified
18	111	Empty	–	–	Null

Some protocols are proposed to improve the identification procedure performance of the QT protocol. These protocols use special technologies, such as bit-tracking [6-11], ID-revising [12-13], and re-identification [14]. Below, we describe these technologies.

- Bit-tracking: In bit-tracking technology, the reader is assumed to have the ability of tracking the transmission bit timing accurately so that the reader can detect which bits are collided when multiple tags transmit their ID bits simultaneously. The BSQTA [6], BSCTTA [6], FQT [7], CTTA [8] protocols and their variants [9-11] adopt the bit-tracking technology for the reader to identify tags. In some of the schemes, tags stop ID responding immediately after bit collision is detected. Those schemes require tags to have the duplex function to send and receive signals simultaneously. Besides, the implementation of BSQTA and BSCTTA also need tags to send responses in time slots one and two, separately. However, it is complex to implement accurate and flexible bit collision detection [15] and the tag sending/receiving duplex function.
- ID-revising: If tag is identified on the basis of the tag ID, the identification procedure performance will be infected by the length and the distribution of the tag ID. In TID QTA [12], a tag ID is replaced by a shorter temporary ID, while QTR [13] reverses the tag ID as the identification ID to change the ID distribution. With the ID-revising technology, the identification procedure performance can be improved, but this technology is suitable only for special ID distribution cases, such as the continuous ID distribution.
- Re-identification: In the re-identification technology, the number of identification iterations is reduced by harvesting the tag ID distribution information from the last identification procedure. For example, AQS [14] adopts such a technology. However, thus technology is just suitable for some cases where the reader needs to repeatedly identify similar sets of tags.

III. THE PROPOSED PROTOCOL

In this section, based on the QT protocol, we propose a stateless and plain tree based anti-collision protocol, called PRQT, by using tag ID partial responses to speed up the tag identification procedure. The proposed protocol adopts the Manchester code for transmitting signals, so even when two or more tags transmit a same bit signal at the same time, the reader can recognize the bit to be 0 (or 1). However, if some tags transmit bit signals of 0 and others transmit bit signals of 1, the reader cannot recognize the bit clearly.

In the proposed PRQT protocol, the reader first sends a request string S of length k, and the tag whose ID prefix matching S responds with its ID postfix of the last $n–k$ bits (i.e., the bits k+1, …,n) to the reader if bit k+1 is "0". However, if bit k+1 is "1", the tag only responds with "1" to the reader. After receiving the first bit responded, the reader can determine one of the four conditions.

C1. If the first bit responded can be recognized as "0" clearly, then the reader can infer that no tag is with ID prefix "S1". Furthermore, if the other bits responded can be received properly, then a tag is identified; otherwise, the reader pushes the request string "S0" into the stack since the reader can infer that two or more tags with ID prefix "S0" respond at the same time.

C2. If the first bit responded can be recognized as "1" clearly, the reader can infer that no tag is with ID prefix "S0" and there is nothing to be received after the first responded bit. The reader just pushes the string "S1" into the stack.

C3. If the first bit responded cannot be identified clearly, the reader can infer that there are tags with ID prefix "S0" and ID prefix "S1". Furthermore, if no other bits are sent out, then the reader just pushes "S1" into the stack. If the other bits responded can be received properly, then a tag is identified and the reader also pushes "S1" into the stack. However, if the other bits responded cannot be received properly, then the reader push "S1" and "S0" into the stack.

C4. If no tag responds, the reader does not push any request string into the stack.

An example of the PRQT protocol execution is depicted in Table II to show the tag interrogation steps 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 7 rounds are explained below. By the example, it is observed that the number of iterations needed to identify all tags is decreased.

Initially: The reader pushes request strings "1" and "0" into the stack.

Round 1: The reader pops up and sends the request string S="0" to tags. The tags with IDs 0001010, 0010101, 0100100 and 0110100 respond since the prefix "0" of the IDs matches with S. The tags with IDs 0001010 and 0010101 send the 6-bit ID postfix to the reader since the first bit responded is "0", while tags with IDs 0100100 and 0110100 just respond with "1" since the first bit responded is "1". In such a case, the first bit responded and the ID postfix cannot be identified properly and the reader pushes request strings "01" and "00" into the stack.

Round 2: The reader pops up and sends the request string S="00" to tags. The tag with ID 0001010 responds with its 5-bit ID postfix, while the tag with ID 0010101 just responds with "1". In such a case, the first responded bit cannot be recognized properly, but the last 4 bits 1010 of the responded ID postfix can. For such a case, the reader can identify the tag with ID 0001010 and knows that tags with ID prefix "S1" are yet to be identified. The request string "001" is pushed into the stack.

Round 3: The reader pops up and send the request string S="001" to tags. Only one tag with ID 0010101 responds and it is thus identified successfully.

Round 4: The request string S="01" is popped up and sent to tags. The tag with ID 0100100 responds with its 5-bit ID postfix, while the tag with ID 0110100 just responds with "1". In such a case, the first responded bit cannot be recognized properly, but the last 5 bits 00100 of the

responded ID postfix can. For such a case, the reader can identify the tag with ID 0100100 and knows that tags with the ID prefix "*S*1" are yet to be identified. The request string "011" is thus pushed into the stack.

Round 5: The request string *S*="011" is popped up and sent to tags. At this time, only one tag with ID 0110100 responds and it is thus identified successfully.

Round 6: The reader pops up and sends the request string *S*="1" to tags. The tags with IDs 1010110 and 1011000 respond with their 6-bit ID postfix, while the tags with IDs 1100111 and 1101010 just respond with "1". In such a case, the first bit responded and the ID postfix cannot be identified properly and the reader pushes request strings "11" and "10" into the stack.

Round 7: The request string *S*="10" is popped up and sent to tags. The tags with IDs 1010110 and 1011000 just respond with "1" at the same time. In such a case, the reader pushes the request string "101" into the stack.

The identification procedure proceeds round by round until the stack is empty. When the stack is empty, all the tags are identified successfully and the identification procedure of the PRQT protocol is finished.

TABLE II. AN EXAMPLE OF THE PRQT PROTOCOL EXECUTION

Round/ Iteration	**Request string *S***	**Stack**	**Response**		**Result**
			Responding tags	***Responding bits***	
Initial	–	0 1	–	–	–
1	0	00 01 1	0001010 0010101	001010 011001	Collision
			0100100 0110100	1 1	Collision at first bit
2	00	001 01 1	0001010	01010	0001010 identified
			0010101	1	Collision at first bit
3	001	01 1	0010101	0101	0010101 identified
			–	–	No collision at first bit
4	01	011 1	0100100	00100	0100100 identified
			0110100	1	Collision at first bit
5	011	1	0110100	0100	0110100 identified
			–	–	No collision at first bit
					collision at first bit
6	1	10 11	1010110 1011000	010110 011000	Collision
			1100111 1101010	1 1	Collision at first bit
7	10	101 11	–	–	Null
			1010110 1011000	1 1	No collision at first bit
8	101	1011 11	1010110	0110	1010110 identified
			1011000	1	Collision at first bit
9	1011	11	1011000	000	1011000 identified
			–	–	No collision at first bit
10	11	110	1100111 1101010	00111 01010	Collision
			–	–	No collision at first bit
11	110	1101	1100111	0111	1100111 identified
			1101010	1	Collision at first bit
12	1101	–	1101010	010	1101010 identified
			–	–	No collision at first bit

IV. SIMULATION

In this section, we show the simulation results of the PRQT protocol and compare them with those of the QT protocol in terms of the number of iterations needed to identify all tags in the interrogation zone. The QT protocol is a stateless and plain protocol. Unlike its variants, such as CTTA, BSCTTA, FQT, BSQTA and BSCTTA, the QT protocol does not incorporate special techniques, like bit-tracking, ID-revising, and re-identification, into the tag identification procedure. The proposed PRQT protocol, based on the QT protocol, is also a stateless and plain protocol. Therefore, we need compare the PRQT protocol with the QT protocol.

The simulation experiments are performed 1000 times for each case of settings and tag IDs are assumed as uniformly distributed. Below, an iteration or a round is defined as a cycle that the reader sends a request string to tags, and 0, 1, or more tags respond, and finally the reader processes the responses.

A. *The number of iterations needed to identify all tags*

This subsection shows the simulation results about the number of iterations needed to identify 100,150, 200, 250,…, 1000 tags within the interrogation zone. It is assumed that tag IDs are uniformly distributed and the length of the tag ID is 64. The simulation results are shown in Fig. 1, by which it can be observed that (1) the number of iterations needed to identify all tags increases with the number of tags, and (2) PQRT is significant better than QT. It is noted that if we divide the number of iterations needed to identify all tags by the total number of tags, we have that QT and PRQT need about 2.88 and 2.44 iterations to identify a tag in average, respectively.

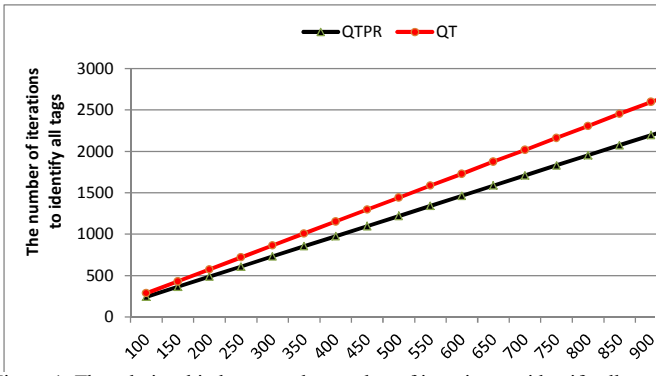
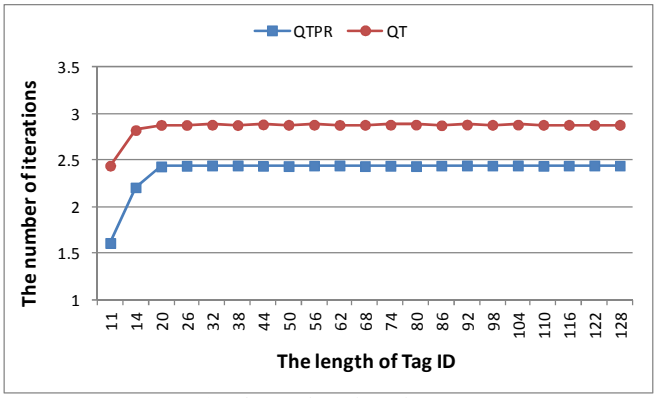


Figure 1. The relationship between the number of iterations to identify all tags and the number of tags

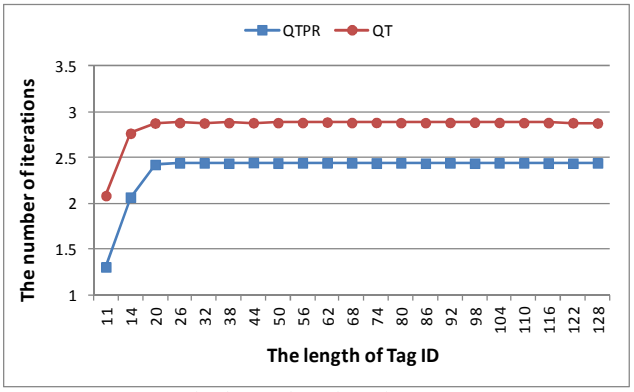
B. *The impact of the ID length on the number of iterations*

In this subsection, we show the influence of the ID length on the number of iterations needed to identify a tag in average for the cases of 500, 1000, 1500 and 2000 tags, and for the cases of 11, 14, 20, 26…, and 128 ID lengths. The simulation results are shown in Fig. 2, by which it can be observed that in the case of tag IDs being distributed sparsely over the ID space (for the ID length larger than or equal to 20), the average number of iterations needed to identify a tag is almost fixed and is about 2.88 and 2.44 for QT and PRQT, respectively. On the contrary, when tag IDs are distributed densely over the tag ID space (for the ID length less than 20), the number of iterations needed to identify a tag in average increases with the ID length. For example, for the case of 1000 tags and the tag ID length of 11, the number of iterations to identify a tag in average is 2.08 and 1.37, respectively. And for the case of 1000 tags and the tag ID length of 14, the number of iterations to identify a tag in average is 2.76 and 2.03, respectively. Again, PRQT is better than QT significantly.

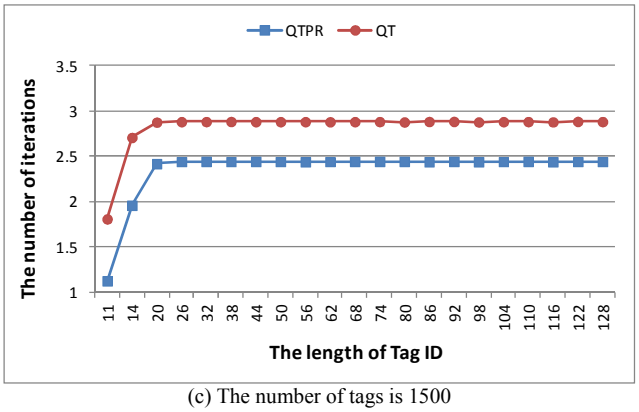
For most cases, tag IDs are distributed sparsely over the ID space. Therefore, we can regard the number of iterations to identify a tag in average is fixed for both PRQT and QT whatever the number of tags is and whatever the ID length is. This is a good property since we can easily estimate the time needed to identify a tag for most cases.



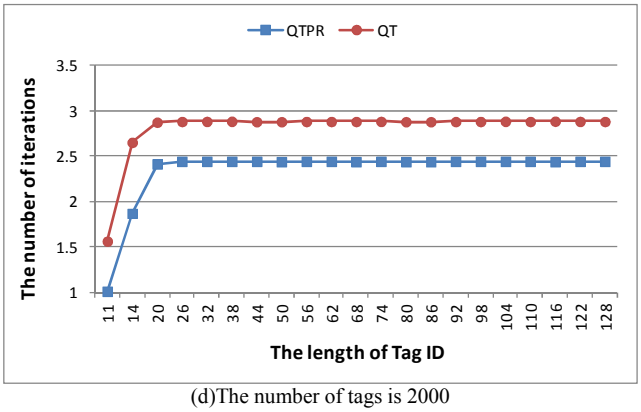
(a) The number of tags is 500



(b)The number of tags is 1000



(c) The number of tags is 1500



(d)The number of tags is 2000

Figure 2. The impact of the tag ID length on the average number of iterations to identify a tag

V. CONCLUSION

This paper proposes a stateless and plain tree-based RFID tag anti-collision protocol, called PRQT, to reduce the number of iterations to identify a tag in average by using tag ID partial responses. The PRQT protocol is nearly as simple as the QT protocol. However, as shown by the simulation results, the number of iterations to identify tags of the former is about 84.7% of those of the latter. This is a significant improvement.

In the future, we plan to conduct the performance comparisons in terms of the number of transmitted bits, the latency, and the energy consumption to identify tags for the QT and PRQT protocols, two plain and stateless tree-based protocols that use no special techniques, such as bit-tracking, ID-revising, and re-identification. We also plan to extend the comparison scope to cover more protocols to make the paper more comprehensive.

REFERENCES

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

[2] Leian Liu and 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.

[3] Ming-Kuei Yeh and 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.

[4] 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 Communications, Aug 2000, pp. 75-84.

[5] 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.

[6] 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.

[7] Gang Wang, Yong Peng, and Zhaomin Zhu, "Anti-collision algorithm for RFID tag identification using fast query tree," IT in Medicine and Education (ITME), 2011 International Symposium on, Volume 1, , 9-11 Dec. 2011, pp.396 - 399

[8] 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.

[9] 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.

[10] Chiu-Kuo Liang and 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.

[11] Xiaolin Jia, Quanyuan Feng, and 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.

[12] 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.

[13] Yun Tian, Gongliang Chen, and 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.

[14] 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.

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