Deterministic Dual-stack Query Tree RFID tag anti-collision algorithm

Yanling Zhou¹, Thomas Clemen²

¹Artificial Intelligence and Big Data College, Hefei University, Hefei 230601, China
²Department of Computer Science, Hamburg University of Applied Sciences, Berliner Tor 7, 20099 Hamburg, Germany

Corresponding author and e-mail: Yanling Zhou, zhouyanling1006@163.com

Abstract. Tag collision problems are a major issue affecting the performance of RFID systems. The probabilistic tag anti-collision algorithm has tag starvation and cannot identify some tags. This paper proposes a deterministic dual-stack query tree algorithm (DDQT). It successfully implements the anti-collision algorithm of tags by generating new query codes and tag identification functions. At the same time, the DDQT algorithm greatly saves memory space by adding a prefix code stack and a suffix code stack. The DDQT algorithm is simple, and there is no limit on the length of the electronic tag to be queried. Compared with the previous query-tree algorithms, the DDQT algorithm has obvious improvements in query comparison times, empty query times, and occupied memory space. It can effectively remove idle nodes, shorten recognition time and save memory resources, thus improving the speed and accuracy of label recognition to some extent.

1. Introduction

RFID (Radio Frequency Identification) technology plays a new type of automatic identification technology. In recent years, it has been widely used at home and abroad, mainly in the following areas, including logistics management, identification, anti-counterfeiting applications, and traffic management. RFID is an automated identification technology that is easy and fast to use. The RFID system is mainly composed of three parts: reader, label and information system. If there are multiple tags in the recognition range of a reader and the tags simultaneously transmit information to the reader, the reader will detect the conflict. The conflict is called "collision" which causes the tag recognition to fail. Tags are widely used in radio frequency identification, and tag collision problems have become a bottleneck limiting the efficiency of RFID systems. The tag collision avoidance problem belongs to the multi-access problem in the communication system [1]. The RFID tag anti-collision algorithm based on time division multiple accesses does not require hardware changes, and can be implemented at the algorithm level. The principle is simple and widely used. Commonly used RFID tag anti-collision algorithms fall into two categories. One is based on the Aloha algorithm, also known as the probabilistic algorithm. Since the number of idle time slots and collision time slots cannot be determined, some tags may not be recognized because of tag starvation. Tree-based algorithms that are also known as deterministic algorithms can avoid starvation and ensure that each tag is accurately identified. Since it is visited according to the binary combination rule, the time of reading the tag is
extended. When the number of tags is large, the efficiency of the algorithm is significantly reduced. The tree-based anti-collision algorithm needs to group the IDs of the tags according to a certain length. When the packet length is 1, it is a binary tree algorithm. When the packet length is 2, it is a quad-tree algorithm. When the packet length is 3, it is the Octree algorithm.

2. RFID tag anti-collision algorithm based on query tree
According to the number of tree branches, it can be divided into binary query tree, quad-check tree, eight-check tree, etc. The binary tree anti-collision algorithm is the most widely used algorithm in RFID system. Its basic principle is: the reader sends a bit of the query code Q (0 or 1 for the tag, which is equivalent to forming two subtrees, first querying the 0 subtree and then querying the 1 subtree). Each tag in the reader response range determines whether its ID starts with Q, and if so, its own ID is transmitted to the reader. At this time, there may be three cases: identification (only one tag starting with Q), collision (two or more tags starting with Q), and idle (five labels begin with Q) [1]. If a collision occurs, add 0 and 1 respectively after the previous query code to form two new query codes (equivalent to splitting into two subtrees). At first, it sends a new query code with 0 added at the end to the label and queries the left subtree. Then it sends a new query code with 1 added at the end to the label and queries the right subtree. If a collision occurs again during the query, the above operation is repeated until all the tags are successfully recognized [2]. In the tree-based anti-collision algorithm, the nodes of the whole tree can be divided into four types: initial node, identification node, collision node and idle node [4]. There are one and only one initial node, and the number of identification nodes is equal to the number of tags. The number of the initial and identification nodes is determined. Therefore, to improve recognition efficiency, the key method is to reduce the number of collision nodes and idle nodes.

In the tree-based RFID anti-collision algorithm, in addition to the binary tree anti-collision algorithm [3,5,6], there is a quadtree anti-collision algorithm [7,8,9]. In the quadtree anti-collision algorithm, the tag group length is 2, and the node coding has four combinations. And a tree with four branches is formed in the querying process. Comparing the quadtree anti-collision algorithm with the binary tree anti-collision algorithm, it can be found that the number of the initial node and the identification nodes are the same, and the number of the collision nodes and the idle nodes are the difference. In the quadtree anti-collision algorithm or hybrid query tree algorithm [7], the number of the collision nodes is less than that in the binary tree collision algorithm, but there are more idle nodes. If the encoding algorithm of the quadtree's query code can be improved and the number of idle nodes is reduced, the recognition efficiency will be greatly improved.

3. The description forming the query code of DDQT algorithm
The DDQT algorithm is a RFID tag anti-collision deterministic dual-stack query tree algorithm. The querying codes are formed by encoding the highest bit and the lowest bit of the collision bit. When collision occurs, it generates four new query codes at a time, removes idle nodes and shortens the recognition time, thereby improving Identify efficiency. The whole algorithm consists of two parts: generating new query code and tag identification. The new query code is generated by the quadtree dual-stack coding algorithm. The identification of the tag is implemented using a quadtree anti-collision algorithm.

The dual-stack algorithm is proposed to solve the problem of memory waste. The code used in this paper is a dual-stack query code, where the query code includes two parts, the plain code and the secret code, as shown in Figure 1 below. When the reader issues the inquiry code, the electronic tag compares with the corresponding code bit according to the plain code in the inquiry code. If the comparing result is the same, the tag sends the entire code to the reader. If the reader can correctly read the electronic tag code, it means there is no collision. When the reader cannot correctly read the electronic tag code, it means that two or more electronic tags are the same as the plain code bits of the query code, and a tag collision occurs.
Figure 1. Example of new query code

When a collision occurs, the reader encodes the highest collision bit and the previous plain code according to the collision bit. The highest bit can take values 0 and 1, generating two new first front half of the plain code. The reader encodes the lowest bit and the following plain code. The lowest collision bit can take values of 0 and 1, generating two new second rear half of the plain code. By means of encoding the high bit and low bit, the two new plain codes of the front half are pushed into the prefix code stack, and the two new plain codes of the rear half are pressed into the suffix code stack. Figure 2 below shows an example of the new contents of the prefix-code stack and the suffix-code stack when the first collision occurred.

![Cipher example](image)

Plain Code

| F | B |
|---|---|
| 10 | 0 |
| 11 | 1 |

Figure 2. Example of prefix-code stack and suffix-code stack

The formation of the new query code is done with a prefix-code stack and a suffix-code stack. The suffix code stack works with the prefix code stack. Since each prefix code corresponds to two suffix codes, the same two suffix codes are used for every two prefix codes. Each time a new query code is generated by combing the prefix code popped from the prefix-code stack and the suffix code popped from the suffix stack.

The two consecutive popped-up suffix codes and the same prefix code are formed two new query codes. The two suffix codes are pushed into the stack again. The next prefix code popped up from the prefix-code stack and the two consecutive popped-up suffix codes is formed two new query codes. The two suffix codes do not need to be pushed into the stack. When the prefix code stack is empty, it means that all tags have been successfully identified. Figure 3 below is a flow chart of the new query code formed by the dual stack.

![Flow chart](image)

Figure 3. Flow chart of forming a new query code by dual stack
The query process of the new query code can include three operations: idle operation, collision operation, and identification tag operation. The idle operation and the identification tag operation are relatively simple. When the collision operation occurs, four new query codes are formed based on the highest and lowest bits of the collision bit. The four query codes are pushed into the stack. Then, continue to pop up from the stacks and form a new query code. Proceed to the next round of inquiry until the stack is empty and the query process is over. At this time, all the tags are identified. In the whole process, all coding operations are done by the reader, thus the reader should have the ability to store and encode. Fortunately, all current readers have these capabilities.

4. Recognizing tags process of DDQT algorithm

In the process of tag identification, the electronic tag may be in three states [4], which are activation state, waiting state, and sleep state.

Activation state: When the reader sends the query code to the tag in the response range, all tags enter the activation state. In addition, when the tag code is consistent with the request code sent by the reader, the tag will also be activated.

Waiting state: When the tag code is different from the query code currently issued by the reader, the tag will temporarily enter the waiting state and wait for the next activation.

Sleep state: When a tag has been identified, it will go to sleep, exit the communication connection, and no longer respond to the reader until the entire recognition process is completed. The specific identification process flow chart is shown in Figure 4 below:

Figure 4. Flow chart of tag identification process

The following is a specific example of the algorithm execution process. Assume that there are 7 tags A, B, C, D, E, F, and G to be identified in the reader response range, and their IDs are 10100011, 11001100, 10101001, 10101100, 100011101, 11001001, 11101001. Here defines the left side of the ID code is high. The identification process is as following Table 1.
Table 1. Identification process of RFID tags under DDQT algorithm

| Times | Query code sent | Query result | Collision code | Prefix code stack | Suffix code stack | New query codes |
|-------|-----------------|--------------|----------------|-------------------|------------------|-----------------|
| 1     | 11111111        | collision    | 1XXXXXXX       | 10 11             | 0 1              | 10XXXXX0 10XXXXX1 11XXXXX0 11XXXXX1 |
| 2     | 10XXXXX0        | Tag D        |                |                   |                  |                 |
| 3     | 10XXXXX1        | collision    | 10X0XXXX1      | 1000 1010         | 01 11            | 1000XXXX0 1000XXXX1 1010XXXX0 1010XXXX1 |
| 4     | 1000XX01        | Tag E        |                |                   |                  |                 |
| 5     | 1000XX11        | NULL         |                |                   |                  |                 |
| 6     | 1010XX01        | Tag C        |                |                   |                  |                 |
| 7     | 1010XX11        | Tag A        |                |                   |                  |                 |
| 8     | 11XXXXX0        | Tag B        |                |                   |                  |                 |
| 9     | 11XXXXX1        | collision    | 11X01001       | 110001001         | 1101001          | 11001001 11101001 |
| 10    | 11001001        | Tag F        |                |                   |                  |                 |
| 11    | 11101001        | Tag G        |                |                   |                  |                 |

According to the case, the execution process of the DDQT algorithm is described in detail.

(1) The query code 11111111 is sent for the first time, and the collision code 1XXXXXXX is obtained. The highest and lowest bits are selected according to the collision code, and then the codes of the highest and lowest bits are 0....0, 0...1, 1.......0, 1..1, the query codes are sequentially formed according to the quadtree dual stack code;

(2) Forming a new query code 10XXXXX0. According to the new query code 10XXXXX0, the tag D can be identified, at which time the RFID and the tag D perform normal communication. The communication ends, and the tag D enters a sleep state;

(3) Forming a new query code 10XXXXX1. According to the new query code 10XXXXX1, a collision occurs and generates a collision code 10X0XXXX1. It will continue to select the highest and lowest bits according to the collision code, and then the code of the highest and lowest bits is 0.....0, 0...1, 1.......0,1.......1, the query code is sequentially formed according to the quadtree dual stack code;

(4) Forming a new query code 1000XX01, according to the new query code 1000XX01, the tag E can be identified. At this time, the RFID and the tag E perform normal communication. The communication ends, and the tag E enters a sleep state;

(5) Forming a new query code 1000XX11, according to the new query code 1000XX11, no response, NULL;

(6) Forming a new query code 1010XX01, according to the new query code 1010XX01, the tag C can be identified. At this time, the RFID and the tag C perform normal communication. The communication ends, and the tag C enters a sleep state;

(7) Forming a new query code 1010XX11, according to the new query code 1010XX11, the tag A can be identified, at this time the RFID and the tag A perform normal communication. The communication ends, and the tag A enters a sleep state;

(8) Forming a new query code 11XXXXX0, according to the new query code 11XXXXX0, the tag B can be identified, at this time the RFID and the tag B perform normal communication. The communication ends, and the tag B enters a sleep state;
(9) Forming a new query code 11XXXXX, according to the new query code 11XXXXX1, a collision occurs, and a new collision code 11X01001 is generated. Since there is only one collision bit, a new search code is formed: 11001001, 11101001, and the prefix stack code is pushed up from the stack. The suffix code stack is empty;

(10) A new query code 11001001 is formed. According to the new query code 11001001, the F tag can be identified. At this time, the RFID and the tag F perform normal communication. The communication ends, and the tag F enters a sleep state;

(11) A new query code 11101001 is formed. According to the new query code 11101001, the tag G can be identified. At this time, the RFID and the tag G perform normal communication, the communication ends, and the tag G enters a sleep state; the entire communication process ends, and all tags are identified.

(12) The prefix code stack is empty and the communication ends.

The query code generated during the whole communication process is as following figure 5. Figure 6 is a change diagram of the dual stack in the producing the query code process and Figure 7 is an explanatory diagram of the symbols of Figure 6.
Figure 7. An explanatory diagram of the symbols of Figure 6

5. Performance analysis and simulation
In the RFID tag anti-collision algorithm, factors affecting performance include the number of query, the number of collisions generated during the query process, the number of the null query generated during the query process, and the size of the memory occupied by the generating query code.

In this paper, seven electronic tags are queried in the reading range of the reader, and four different tag anti-collision algorithms are compared. The four anti-collision algorithms include binary query tree, quadtree, hybrid query tree, and dual-stack query tree. The influencing factors of the four algorithms are shown in Table 2 below.

Table 2. Comparison of performance parameters of different anti-collision algorithms

| Query tree type | Query Number | Collision Number | Empty Query Number | Memory Size (bit) |
|-----------------|--------------|------------------|--------------------|------------------|
| Binary Tree     | 18           | 7                | 3                  | 68               |
| Quad Tree       | 11           | 3                | 1                  | 52               |
| Hybrid Query Tree | 11          | 3                | 0                  | 48               |
| Dual Stack Query Tree | 11        | 3                | 1                  | 34               |

In the RFID query tree tag anti-collision algorithm, the memory space is mainly affected by the number of search codes, and the number of search codes is affected by the number of collisions. When the number of collisions is more, the number of search codes generated is larger. According to the analysis, the number of collisions of the binary query tree is twice of the quadtree query tree. However, the binary query tree generates two query codes in the process of one collision, and the quad-tree query algorithm generates four query codes in the process of one collision. Therefore, there is no significant improvement in the occupation size of the memory space. The DDQT algorithm combines different pop-up codes into different query codes. In the process of collision, the two parts of the query code are stored in the prefix-code stack and the suffix-code stacks in a separate manner to achieve different prefix and suffix codes to store in the same memory space. In the DDQT algorithm, the generation of the query code is dynamic. The process of the traditional query tree code generation is a static process, and the query codes always occupy memory space after generation. By comparison, the DDQT algorithm greatly improves the utilization of memory space compared to the other three algorithms.
6. Conclusion

The DDQT algorithm greatly saves memory space by adding a prefix-code stack and a suffix-code stack. And the algorithm is simple and easy to implement. There is no limit to the length of the electronic tag which needs to query. Compared with the binary query tree algorithm, quadtree query tree algorithm and hybrid query tree algorithm, the querying number of DDQT algorithm is the same as that of the quadtree query tree algorithm and the hybrid query tree algorithm, which is nearly half of that of the binary query tree algorithm. The number of collisions is the same as that of the quadtree query tree algorithm and the hybrid query tree algorithm, which is significantly less than that of the binary query tree algorithm. On the null query, it is better than the binary query tree algorithm, which is not much different from the quadtree query tree algorithm and the hybrid query tree algorithm. In occupation of the memory space, it is obviously smaller than the other three algorithms. The binary tree anti-collision algorithm has the largest memory space occupancy rate, which is twice of the DDQT algorithm. The memory-occupancy rate of DDQT algorithm is significantly reduced by half compared to the binary tree query algorithm and significantly improved compared to the quadtree query tree algorithm and the hybrid query tree algorithm.

Acknowledgments

This paper is supported by the Excellent Talents Training Funded Project of Universities of Anhui Province (No. gxgwfx2019065) and Provincial Humanities and Social Science Research Project of Anhui Colleges (No.SK2018A0605).

References

[1] M.K. Yeh, Y.L. Lai, J.R. Jiang. An Efficient Query Tree Protocol for RFID Tag Anti-Collision[C]. IEEE 2015: 346-352.
[2] Kuo-Hui Yeh, N.W. Lo. An efficient tree-based tag identification protocol for RFID systems[C]. 22nd International Conference on Advanced Information Networking and Applications, WAINA, 2008, 220: 966-970.
[3] Yun Tian, Gongliang Chen, Jianhua Li. Bi-slotted binary tree algorithm with stack for radio frequency identification tag anti-collision[J]. Journal of Shanghai Jiaotong University (Science), 2013, 18(2): 173-179.
[4] Wu J. Henxin J and etc. An improved Query Tree Anti-collision Algorithm in RFID System [J]. Computer technology and development. 2015, 25(2): 86-89.
[5] Zhiguo X. Yigang H. Zhouguo H. Binary-tree bit-detecting RFID tag anti-collision algorithm[J]. Computer Engineering and Applications, 2010, 46(20): 245-248.
[6] T. Wang. Enhanced binary search with cut-through operation for anti-collision in RFID systems. IEEE Commun. Lett., Vol.10, No.4, April 2006: 236-238.
[7] J. Ryu, H. Lee, Y. Seok etc. A Hybrid Query Tree Protocol for Tag Collision Arbitration in RFID system. IEEE ICC, Jun 2007: 981-986.
[8] Y. Kim, S. Kim, S. Lee and K. Ahn. Improved 4-ary Query Tree Algorithm for Anti-Collision in RFID System[C]. IEEE Computer Science. 2009: 699-704.
[9] T.F. Shin, C.H. Teng. The Design of a Stride Query Tree Algorithm for RFID Systems[C]. IEEE computer Science. 2011: 108-113.