Research Article

A Novel Hybrid Anticollision Algorithm for RFID System Based on Grouped Dynamic Framed Recognition and Binary Tree Recursive Process

Jian Yang, Yonghua Wang, Qingling Cai, and Yiju Zhan

1Faculty of Automation, Guangdong University of Technology, Guangzhou 510006, China
2School of Engineering, Sun Yat-sen University, Guangzhou 510006, China

Correspondence should be addressed to Jian Yang; yangjianijdsn@163.com

Received 31 October 2014; Revised 23 November 2014; Accepted 8 December 2014

Copyright © 2015 Jian Yang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recently, RFID technology has come into end-user applications for monitoring, tracking, and so forth. In RFID system, a reader identifies a set of tags over a shared wireless channel. When multiple tags communicate with the same reader simultaneously, all packages will be lost and no tag can be recognized, which is known as tag collision. Tag collision is a significantly important issue for fast tag identification. We firstly make a thorough analysis among a variety of traditional RFID anticollision algorithms. Then a novel hybrid anticollision algorithm called T-GDFSA is proposed. Tags are assigned to different groups based on the initial tag estimation and then experience several dynamic read frames for identification. When a collision occurs in current slot, a tree-based recursive process will be deployed immediately. T-GDFSA combines the advantages of ALOHA-based and tree-based together and acquires higher system throughput by reducing unnecessary idle and collision slots and lower communication complexity by decreasing the data transmitted, which makes it identify tags faster with less power consumption. Simulations show that the theoretical values match well the simulation results. Moreover, T-GDFSA also has a good tolerance for the inaccuracy of initial tag estimation and the length variation of tag's ID.

1. Introduction

In recent decades, Radio Frequency Identification (RFID) has become one of the most commonly used technologies in our daily life. RFID technology is similar to the technologies of barcodes and magnetic strips, but, in addition, it has the capability of identifying objects uniquely and even without requiring direct line of sight. It is nowadays getting more prevalent in daily applications, such as electronic toll collection, public transport, access control and security, and airport luggage position tracking [1, 2]. A basic RFID system is composed of a reader and a set of tags. The reader inquires tags that are able to communicate on the wireless channel, returning their IDs. Tags are typically passive devices, which answer to reader's query by backscattering the received signal [3, 4]. However, because of all readers and tags sharing the same communication channel, when more than one reader tries to transmit to the same tag or more than one tag tries to communicate with the same reader simultaneously, all their packages will be lost. This phenomenon is called RFID system collision, which leads to long identification time, low system efficiency, and high power consumption [5, 6]. As illustrated in Figure 1, there are two different categories of collisions in RFID system: (1) reader collision and (2) tag collision. Reader collision happens when multiple readers coexisting in the same area interfere with each other, and can be overcome by the coordination strategy among readers. Tag collision occurs among tags simultaneously transmitting to the same reader and may be more difficult to solve because of constraints of energy supply and computing complexity on tags with low functionality. Since tag collision is considered as a key problem affecting the universal adoption of the RFID system, in this paper, we primarily focus on tag collision in single reader scenarios.

In order to avoid tag collision, many tag anticollision algorithms have been proposed so far, which can be classified
into two categories, ALOHA-based probabilistic anticollision algorithms [7–10] and tree-based deterministic anticollision algorithms [11–13]. ALOHA-based algorithms decrease the probability of collision by randomly scheduling the response time of tags. In those algorithms, identification time is slotted and time slots form up read frames. Frame size varies over time and is broadcast by the reader to all tags at the beginning of each read frame. When receiving frame size, tags select one time slot from current read frame randomly and only respond the reader when the selected time slot comes. Colliding tags are scheduled into the next read frame to be slotted and identified, until there are no tags left. The main representatives are DFSA [8, 9] and EDFSA [10], both of which need to estimate the initial tag population to set the proper read frame size to optimize performance. However, when the number of tags becomes much larger compared with the frame size, the performance deteriorates seriously. A detailed study of these two algorithms is presented in Section 2. On the other hand, tree-based algorithms proceed with the identification process represented as a tree where the root responds a set of tags to be identified. The algorithms periodically query a subset of tags which match the given query property, and continuously split a set of tags into two subsets until each set has only one tag. Although the identification time may be longer than that of ALOHA-based algorithms, they are deterministic which can avoid that a specific tag may not be identified all the time, leading to the so-called "tag starvation problem." The main representatives are BS [11], QT [12], and QTI [12]. Unfortunately, there are many distinct disadvantages in BS, QT, and QTI. A great number of colliding subsets exist in BS, while performance of QT and QTI is very sensitive to the distribution of tag’s ID. A more detailed description is shown in Section 2.

Furthermore, a good tag anticollision algorithm for RFID passive tags should have the following characteristics. Firstly, a reader must identify all tags existing in its covering range, as tag starvation problem may cause failure of tracking and positioning objects. Second, system throughput should be high enough to ensure tags are recognized promptly, which is of great importance for mobile tag identification. Finally, communication complexity ought to be low, since passive tags supplement power supply from the reader’s electromagnetic wave, which causes the available power for tags being very limited. On the other hand, low communication complexity is also very useful for the RFID applications where large scale mobile readers are deployed.

Based on the analysis hereinbefore, we propose a novel hybrid anticollision algorithm to improve the performance of RFID tag identification. The new algorithm, called Grouped Dynamic Framed Recognition and Binary Tree Recursive Process (T-GDFSA), generally identifies tags by grouped dynamic frames and calls binary tree recursive process to solve each colliding slot. As T-GDFSA combines ALOHA-based and tree-based anticollision together, it avoids the waste of idle slots in the late identification stage in ALOHA-based and colliding slots in the early identification stage in tree-based. Also because of the combination, the communications between the reader and tags become reduced which means less power is consumed. According to the later simulations in Section 4, T-GDFSA acquires higher system throughput by reducing unnecessary idle and collision time slots and lower communication complexity by decreasing the data transmitted between the reader and tags, which overwhelm other main anticollisions and make it easier to be realized in the end-user applications.

The remainder of this paper is organized as follows. In Section 2, a variety of ALOHA-based and tree-based tag anticollision algorithms are observed specifically. In Section 3, the main concept and description of T-GDFSA are present in detail. In Section 4, evaluation of time complexity and communication complexity of T-GDFSA are proposed, respectively. In Section 5, performance simulations between T-GDFSA and other anticollision algorithms are shown via Matlab. Lastly, Section 6 gives conclusion.

2. RFID Tag Anticollision Algorithms

In this section, we review the main representatives of ALOHA-based and tree-based tag anticollision algorithms for RFID system. The major metrics observed include time
2.1. Dynamic Framed Slotted ALOHA (DFSA). DFSA is an ALOHA-based algorithm which can adjust the read frame size dynamically according to the estimation of tag population. The identification process is made of several read frames, which is slotted into intervals of time, called time slot, whose duration is equal to the tag’s ID transmission time. Before the initialization of the first read frame, the reader estimates the unidentified tag population \( n \) and sets the read frame size \( L \). Tags randomly select one time slot in the range of \([1, L]\) and only respond the reader when the selected slot comes. As the proper frame size is decided in advance, DFSA decreases the potential collision probability to a great extent. The population of unidentified tags is estimated in the following way. Let \( N_c \) be the number of colliding time slots; let \( N_r \) be the number of time slots with single tag which can be identified successfully and let \( C_{\text{ratio}} \) be the ratio of \( N_c \) and \( n \) which can be calculated by (1). When the current read frame is finished, the population of unidentified tags varies to \( N_c/C_{\text{ratio}} - N_r \), which is set to be the new size of the next read frame to maintain an optimal system throughput:

\[
C_{\text{ratio}} = 1 - \left( 1 - \frac{1}{L} \right)^{n} \cdot \left[ 1 + \frac{n}{L-1} \right].
\] (1)

If we define \( T_{\text{slot}} \) as the duration of time slot, \( L_{\text{ini}} \) as the initial frame size, \( L \) as the actual frame size after initial frame, \( P_{\text{succ}}(k) \) as the possibility of successful identification for a tag after \( k \) times attempts, \( Q_{\text{Rep}} \) as the symbol length of slot interval, \( A_{\text{ack}} \) as the symbol length of acknowledgment sent by the reader after receiving tag’s ID, and \( ID_{\text{bits}} \) as the length of tag’s ID, the major metrics of DFSA can be described as follows:

\[
T = T_{\text{slot}} \cdot \left( L_{\text{ini}} + L \cdot \sum_{k=1}^{\infty} k P_{\text{succ}}(k) \right),
\]

\[
\text{Bits}_{R} = L_{\text{bits}} \cdot \left( \sum_{k=1}^{\infty} k P_{\text{succ}}(k) + 1 \right) + Q_{\text{Rep}} \cdot L
\]

\[
\text{Bits}_{T} = \text{ID}_{\text{bits}} \cdot \sum_{k=1}^{\infty} k P_{\text{succ}}(k).
\]

Table 1: The strategy of grouping and frame sizing.

| Tag population \( n \)   | Frame size \( L \) | Group size \( G \) | Tag population \( n \)   | Frame size \( L \) | Group size \( G \) |
|--------------------------|---------------------|---------------------|--------------------------|---------------------|---------------------|
| ...                      | ...                 | ...                 | 177–354                  | 256                 | 1                   |
| 6–11                     | 8                   | 1                   | 355–707                  | 256                 | 2                   |
| 12–19                    | 16                  | 1                   | 708–1416                 | 256                 | 4                   |
| 20–40                    | 32                  | 1                   | 1417–2831                | 256                 | 8                   |
| 41–81                    | 64                  | 1                   | ...                     | ...                 | ...                 |
| 82–176                   | 128                 | 1                   |                           |                     |                     |

However, when tag population is large, the performance of DFSA deteriorates heavily. This is because the size of memory recording the selected slot in tags is limited and cannot keep pace with the increase of tag population, which makes the optimal condition that the frame size is set equal to the tag population not satisfied.

2.2. Enhanced Dynamic Framed Slotted ALOHA (EDFSA). On the basis of DFSA, in order to enhance the system performance when tag population is large, EDFSA introduces the grouping strategy that it randomly splits tags into different groups according to the maximal frame size and the estimated number of unread tags. Only the tags associated with one of the groups are queried in the following read frames; then the identifications for tags in other groups are followed sequentially. In fact, the grouped identification is more like a combination of several DFSA processes. Chebysev’s inequality is used in the end of each read frame to estimate the number of tags left, which is used to adjust the number of groups and frame size for the incoming read frames.

The grouping and frame sizing strategy is represented in Table 1, where the group number and frame size are adjusted in the way of doubling or halving, which can be easily realized in practical applications by the operation of left shifting or right shifting the group register.

If we define the same notations as given in DFSA above, the major metrics of EDFSA can be derived as follows:

\[
T = T_{\text{slot}} \cdot \left( G_{\text{ini}} \cdot L_{\text{ini}} + G \cdot L \cdot \sum_{k=1}^{\infty} k P_{\text{succ}}(k) \right),
\]

\[
\text{Bits}_{R} = G_{\text{bits}} \cdot L_{\text{bits}} \cdot \left( \sum_{k=1}^{\infty} k P_{\text{succ}}(k) + 1 \right) + Q_{\text{Rep}} \cdot L \cdot \sum_{k=1}^{\infty} k P_{\text{succ}}(k)
\]

\[
\text{Bits}_{T} = \text{ID}_{\text{bits}} \cdot \sum_{k=1}^{\infty} k P_{\text{succ}}(k).
\]
2.3. Binary Tree Algorithm (BS). BS and Query Tree (QT) are both tree-based algorithms, in which the reader iteratively queries a subset of tags matching a given property until all tags are identified. They differ in the way of querying tags: according to a binary random number in BS and a query prefix in QT. In BS, the reader recognizes tags in the time interval called read cycle. Colliding tags are split into two subsets recursively until just one tag is allocated in every subset. The tag has a counter initialized to 0 in the very beginning. Only tags with counter value equal to 0 can respond the reader query in current read cycle, and other tags keep silent and wait for their counters decreasing to 0. Tags modify the value of counter depending on the result of current identification. If a collision happens, colliding tags add a 1-bit binary value randomly to their counters, which are split into two subsets with the counter at 0 and the counter at 1, respectively. Other tags not involved in the collision increase their counters by 1 to maintain their former identifying order unchanged. In case of identifying which are split into two subsets with the counter at 0 and of current identification. If a collision happens, the reader uses two 1-bit longer queries, that is, \( q_1q_2\cdots q_n \) and \( q_1q_2\cdots q_n1 \), in the next read cycles in turn, which splits the set of colliding tags into two subsets. If a collision still occurs when the new 1-bit longer query string is sent, then the string length is further increased by 1 bit with 0 or 1 added to the end of the current query string, until the collision is solved and a tag is identified successfully. The reader has a stack \( S \) to manage query strings. In the beginning, \( S \) is initialized as NULL, which causes all tags response. When collision occurs, the reader pushed two 1-bit longer query strings into \( S \). When current read cycle is finished, it pops a query string from \( S \) for the next read cycle. That \( S \) is popped empty indicates that all tags are recognized. Notice that collisions in QT correspond to the intermediate nodes, not including the root node, compared with BS.

If we define the same notations as given in BS above, the major metrics of BS can be expressed as follows:

\[
T = T_{cycle} \cdot \left(1 + 2 \cdot \sum_{k=0}^{\infty} \left(2 - (2^k + n - 1) \cdot \left(1 - 2^{-k}\right)^{n-1}\right)\right),
\]

Bits\(_R\) = \( C_{bits} \cdot \left(1 + 2 \cdot \sum_{k=0}^{\infty} \left(2 - (2^k + n - 1) \cdot \left(1 - 2^{-k}\right)^{n-1}\right)\right)\),

\[
\text{Bits}_T = \text{ID}_{bits} \cdot \log_2 n.
\]

The expressions (4) and (5) are general equations when \( k \), collision times, that is, the depth of current node (read cycle) in the binary tree (the whole identification process), tends to positive infinity. For convenience, we derive the lower bounds of \( T \) and Bits\(_R\), for reference. As in BS the binary tree representing the whole identification process is a strict binary tree; the intermediate nodes denoted by \( C_p(n) \) (i.e., the colliding times) achieve minimal value when they are regarded as a complete binary tree whose number of nodes can be expressed as \( 2C_p(n) + 1 \). Consider the fact that the number of all nodes in the strict binary tree satisfies the inequation \( N_{nd}(n) \geq 2C_p(n) + 1 = 2 \cdot \sum_{k=0}^{\log_2 n - 1} 2^k + 1 = 2n - 3 \), and each node corresponds with a read cycle, so we can derive the following expressions for fast tag estimation:

\[
T \geq T_{cycle} \cdot (2n - 3),
\]

\[
\text{Bits}_R \geq \text{ID}_{bits} \cdot (2n - 3).
\]

2.4. Query Tree Algorithm (QT). QT splits a tag set according to tag’s ID. The reader transmits a query including a bit string, and only the tag whose ID has a prefix matching the query string responds by transmitting its ID. In particular, if the current query string is \( q_1q_2\cdots q_n \) (\( q_i \in \{0,1\} \)) and leads to a collision, the reader uses two 1-bit longer queries, that is, \( q_1q_2\cdots q_n0 \) and \( q_1q_2\cdots q_n1 \), in the next read cycles in turn, which splits the set of colliding tags into two subsets. If a collision still occurs when the new 1-bit longer query string is sent, then the string length is further increased by 1 bit with 0 or 1 added to the end of the current query string, until the collision is solved and a tag is identified successfully. The reader has a stack \( S \) to manage query strings. In the beginning, \( S \) is initialized as NULL, which causes all tags response. When collision occurs, the reader pushed two 1-bit longer query strings into \( S \). When current read cycle is finished, it pops a query string from \( S \) for the next read cycle. That \( S \) is popped empty indicates that all tags are recognized. Notice that collisions in QT correspond to the intermediate nodes, not including the root node, compared with BS.

If we define the same notations as given in BS above, the major metrics of QT can be derived as follows:

\[
T \leq T_{cycle} \cdot (2.887n - 1),
\]

\[
\text{Bits}_R \leq \text{ID}_{bits} \cdot (2.887n - 1),
\]

\[
\text{Bits}_T = \text{ID}_{bits} \cdot (2.21\log_2 n + 4.19).
\]

2.5. Query Tree Improved Algorithm (QTI). There are amounts of useless query strings in QT, which increase the read cycles and depress the system throughput. QTI is an enhancement that removes those certainly causing collisions in advance. For example, consider the case in which query prefix \( q \) has produced a collision, while \( q0 \) result in no response. As \( q1 \) definitely leads to a collision, QTI skips it directly and takes \( q10 \) as the next query prefix.

Compared with QT, QTI only reduces the amount of data transmitted by the reader, while it maintains the amount of data sent by tags unchanged, which also makes the expression of \( \text{Bits}_T \) unchanged.

If we follow the same notations defined in BS, we can obtain the formulas as follows:

\[
T \leq T_{cycle} \cdot (2.665n - 1),
\]

\[
\text{Bits}_R \leq \text{ID}_{bits} \cdot (2.665n - 1),
\]

\[
\text{Bits}_T = \text{ID}_{bits} \cdot (2.21\log_2 n + 4.19).
\]

From what has been discussed hereinbefore, we can see clearly that, regardless of time or communication complexity, the metrics of ALOHA-based, including DFSA and EDFSA,
are the same order infinitesimal of \( n \), while the metrics of tree-based, including BS, QT, and QT1, are the higher order infinitesimal of \( n \). Although the metrics of tree-based algorithms seem worse, they are deterministic which can avoid the so-called "tag starvation problem" that a specific tag may not be identified all the time in ALOHA-based algorithms.

3. The Proposed Algorithm T-GDFSA

The proposed algorithm T-GDFSA can be divided into two steps, the step of the estimation of tag population and the step of tag identification. The former is just implemented once to estimate the initial number of tags existing in the covering range of the reader, which is used in the second step to calculate the optimal number of groups and frame size in each group. In T-GDFSA, the method used for tag estimation is similar to TEM (Tag Estimation Method) [14]. At the end of each read cycle, the recorded triple numbers \( \langle c_0, c_1, c_k \rangle \) can be obtained from the reader’s memory, which quantifies the number of idle time slots, successful time slots occupied by only one tag, and colliding slots with more than two tags, respectively. Meanwhile, the expected triple numbers \( \langle a_0^{N,n}, a_1^{N,n}, a_k^{N,n} \rangle \) can be computed from the given frame size \( N \) and the number \( n \) of tags by statistical knowledge.

According to Chebyshev’s expression, the outcome of a random experiment involving a random variable \( X \) is most likely somewhere near the expected value of \( X \). This property is used to calculate the distance between the recorded results \( \langle c_0, c_1, c_k \rangle \) and the expected results \( \langle a_0^{N,n}, a_1^{N,n}, a_k^{N,n} \rangle \) of each read cycle as defined in (10). When the minimal distance is achieved, the corresponding \( n \) is considered to be the optimal estimation of tag population. Consider

\[
\varepsilon (N, c_0, c_1, c_k) = \min_n \left( \left| \frac{c_0}{a_0^{N,n}} - \frac{c_1}{a_1^{N,n}} - \frac{c_k}{a_k^{N,n}} \right| \right), \quad (10)
\]

where \( a_k^{N,n} \) is the expected value of colliding slots with \( k \) tags; then

\[
a_k^{N,n} = N \cdot \left( \begin{array}{c} n \end{array} \right) \cdot \left( \frac{1}{N} \right)^k \cdot \left( 1 - \frac{1}{N} \right)^{n-k} \quad (k \leq n). \quad (11)
\]

In (10), the minimal \( \varepsilon \) is supposed to be achieved by a searching process with \( n \) varying in the range \([c_1 + 2c_k, 3(c_1 + 2c_k)]\), where \( c_1 + 2c_k \) is the lower bound of \( n \) according to [15], since \( c_k \) tags have been identified, and if there are \( c_k \) collisions, at least \( 2c_k \) tags collided. The upper bound of \( n \) is set to \( 3(c_1 + 2c_k) \), since by simulation no further accuracy in the estimation is obtained if the upper bound is set to higher values.

From [1] we know that the system throughput becomes optimal when the frame size is set equal to the number of tags. In order to satisfy this condition, the grouping size is adjusted by 1 each time to ensure that the number of tags in each group is very close to the frame size. The optimal grouping size is decided by the expected system efficiency. Suppose the current number of groups is \( k \); it will be increased to \( k + 1 \) if the expected system efficiency with \( k + 1 \) groups rises just equal to that with \( k \) groups. Consider

\[
a_k^{N,n(k)}/N \leq a_k^{N,n(k+1)}/N \quad (k = 1, 2, \ldots). \quad (12)
\]

From (12), the maximum number of tags with \( k \) groups is

\[
n_k = \frac{\lg (k/(k + 1))}{\lg (1 - 1/N)} \cdot k \cdot (k + 1) \quad (k = 1, 2, \ldots). \quad (13)
\]

By the utilization of (13), the optimal grouping size \( G \) can be obtained with a varying \( n \) and a given frame size \( L \). Then the optimal \( G \) is broadcast by the reader with a random number to tags. Grouping rule is as follows. Each tag generates a new random number by the old received one and its serial number (tag's ID). Then the new random number is divided by \( G \), and all tags have a group number randomly in the range \([0, G - 1]\). In the current read frame, only tags with group number 0 are permitted to respond the reader. When all of them have been identified, the reader begins a new read frame and tags with group number 1 are involved. The process continues until tags having group number \( G - 1 \) are queried, which means all tags have been recognized. The frame size of each read frame decides how many time slots are included in it. At the beginning of each read frame, the reader broadcasts the optimal frame size \( L \) to all tags which means there are \( L \) time slots in the incoming read frame, and tags involved generate an integer \( L_i \) in the range \([1, L]\) and save it in its slot counter. By this means, tags are allocated into \( L \) time slots randomly. In the first time slot initiated by the reader, tags with slot counter at 0 are permitted to respond the reader. When all tags in current time slot have been identified, the reader polls the next time slot and informs other unidentified tags to decrease their time slot counter by 1. Tags whose value of slot counters decreasing to 0 are permitted to respond the reader in the incoming time slot. If a collision happens in the current time slot, tags involved add 0 or 1 randomly to their slot counters which cause them to fall into two subsets. It means that a binary tree is introduced, and colliding tags are subdivided into two child nodes of the current parent node randomly. Other tags increase their slot counters by 1 so as to counteract the effect caused by splitting for the colliding tags to keep their former identification sequence unchanged. After splitting, the reader firstly turns to identify the tags in the left child node of current parent node, and if a collision is still detected, the node-splitting process will be executed again and tags falling into the deeper left child node will be read in the incoming time slot. The process will keep on going recursively until there is no collision happened in the current left child node. Because there is no more than 1 tag in the leaf, tag in it can be identified successfully and then muted. Next, the reader returns up to the parent node of the current left child node, then turns down to the right child node and identifies tags in it. There are three possible outcomes. Collision slot will cause a new node-splitting process similar to that represented beforehand. 1-tag slot will make the reader identify the tag successfully and then return to the parent node. Idle slot will make the reader directly come back to the parent node. Next,
as all the child nodes of the parent node have been identified, the reader will return to the upper parent node. The procedure will continue recursively until all nodes of the binary tree have been queried, which means all tags in current colliding time slot have been identified. In the initiation of each binary tree recursive process, the reader sets a level counter for current colliding slot to record the depth of current node (i.e., read cycle) in the binary tree. It is increased by 1 when a collision happens in current node and the reader goes deeper to the child nodes and is decreased by 1 when all tags in the child nodes are already identified and the reader returns to the upper parent node. Level counter which decreased to 0 indicates that all tags in the current collision slot have been identified. In Figure 2, the tag identification process for group 1 is illustrated, where Lev means the level counter, \( \Phi \) denotes no tag responding, and each “•” represents one tag. The identification processes for other groups are similar.

4. Performance Evaluation

In this section we will evaluate the performance of T-GDFSA. There are several metrics focusing on different aspects of RFID system. We take time complexity and communication complexity for emphatic analysis. Because time complexity reflects the identification efficiency which is of great importance for fast tag identification, and communication complexity indicates power consumption of the reader and tags which is of great significance for mobile RFID applications. Time complexity is defined as the total time spent to identify a set of tags in the covering range of the reader. Communication complexity refers to the amount of data transmitted between the reader and tags, which can be composed of two parts: \( \text{Bits}_R \) (the total number of bits sent by the reader) and \( \text{Bits}_T \) (the total number of bits transmitted by single tag in average).

Prior to deeper analysis, it is necessary to make the following important assumptions.

**Assumption 1.** As data received by the reader are mostly short symbols such as acknowledgement symbols and so forth, which are much less compared with the data sent by the reader, we simply ignore it and consider that \( \text{Bits}_R \) totally represents the communication complexity on reader.

**Assumption 2.** The duration of time slot in read frame is set equal to that of read cycle in binary tree, which is denoted by the unified symbol \( T_{\text{cycle}} \), making it possible to calculate the system throughput quantificationally.

**Assumption 3.** In most RFID applications, power consumption of tags is more sensitive than that of the reader. So we proposed a new strategy dedicated to reducing \( \text{Bits}_T \), which is deployed as follows. Instead of sending complete tag’s ID directly, tag firstly sends IDTest (a short bit string) to test the state of the current time slot. If no collision happens, IDTest can be received by the reader successfully. The reader responds with the acknowledgement IDReq to require the tag to transmit its complete ID. The strategy eliminates the useless mass data transmission of tag’s ID caused by collisions and ensures that tag’s ID is sent only once. Since the length of IDTest is much shorter than that of tag’s ID, \( \text{Bits}_T \) is decreased drastically.

4.1. Time Complexity. The total consumed time, denoted by \( T \), consists of three parts, \( T_{\text{est}} \) (time cost in the estimation of tag population), \( T_{\text{rf}} \) (time cost in read frames), and \( T_{\text{rc}} \) (time cost in read cycles of binary tree recursive process). Then we get

\[
T = T_{\text{est}} + T_{\text{rf}} + T_{\text{rc}}.
\]  (14)

Let \( G_{\text{ini}} \) denote the initial grouping size for the estimation of tag population, and let \( G(n) \) denote the actual grouping size calculated by (13). \( L_{\text{ini}} \) and \( L(n) \) represent the initial frame size and the actual frame size separately. \( n \) is the number of unidentified tags. Then \( T_{\text{est}} \) and \( T_{\text{rf}} \) can be expressed as \( G_{\text{ini}} L_{\text{ini}} T_{\text{cycle}} \) and \( G(n) L(n) T_{\text{cycle}} \), respectively. According to [1], when the condition of maximal throughput of framed slotted ALOHA, the read frame size is equal to the number of unidentified tags, is satisfied, the average number of tags in...
each colliding time slot is about 2.3922 on average. Because $G(n)L(n)$ is very close to $n$ in T-GDFSA, the average number of tags in each colliding time slot, denoted by $C_{tag}$, can be considered as 2.3922 simply.

According to the representation above, we know that the binary tree process for each colliding time slot is a strictly binary tree, so there is no 1-degree node included, meaning that any non-leaf node (the root node or the intermediate node) has two child nodes. Let $N_{tag}$ and $N_{leaf}$ be the number of non-leaf nodes and leaf nodes, respectively; then $N_{tag} = 2N_{leaf} - 1$ can be easily deduced [16]. Since the root node corresponds to the colliding time slot which has already been counted in the read frame before, the effective number of nodes in every binary tree is $N_{nd} - 1$. Since the leaf nodes are composed of 1-tag leaf nodes and idle leaf nodes, the number of actual nodes in every colliding time slot, denoted as $N_{nd}$, satisfies the inequality $N_{nd} \geq 2C_{tag} - 2 = 2.7844$ on average. Note the fact that when $C_{tag}$ is small it is very close to $N_{leaf}$, so the inequality above can be considered as $N_{nd} = 2C_{tag} - 2 = 2.7844$ simply. By probability knowledge, we can derive that the expected value of collision times slots is $G(n)L(n)P_{coll}$, where $P_{coll}$ is the probability of detecting a colliding time slot, and that $T_{rc}$ is no less than $2.7844G(n)L(n)P_{coll}T_{cycle}$. Substituting (14) with all results above, we acquire the final expression of $T$ as follows:

$$T \approx \left[ G_{ini} \cdot L_{ini} + G(n) \cdot L(n) + 2.7844 \cdot G(n) \cdot L(n) \right] \cdot P_{coll} \cdot T_{cycle}.$$  (15)

In expression (15), $P_{coll} = \left[ 1 - n/G(n) \cdot p(1 - p)^{n/G(n)-1} - (1 - p)^{n/G(n)} \right]$ where $p = 1/L(n)$ is the probability of a tag selecting a time slot from total $L(n)$ time slots.

4.2. Communication Complexity on Reader. $Bits_R$ is defined as the amount of data transmitted by the reader to identify a set of tags appearing in its irrigation range, which reflects the communication complexity on reader suitably. From the above analysis, we realize that, no matter in the step of initial tag estimation or in the step of tag identification process, the amount of data sent by the reader representing group size and frame size are both $(G_{bits} + L_{bits})$. The reader also needs to transmit QueryReq periodically to make the current time slot increased continually so as to poll tags allocated in different time slots of current frame, and the amount of data of this part can be expressed as $[G_{ini}L_{ini} + G(n)L(n)]QueryRep_{bits}$. Besides, in every binary tree recursive process, as expressed above, the number of read cycles, $N_{rc}$, is equal to $2C_{tag} - 2 = 2.7844$. As nodes in a binary tree consist of collision nodes and leaf nodes and the leaf nodes need no extra read cycles to query, the number of actual nodes for which reader sends read cycles should be $(2C_{tag} - 2 - C_{tag}) = 0.3922$ on average. The reader sends at most 1 read cycle in every node, so the number of read cycles in every binary tree recursive process is almost 0.3922. Considering that the expected number of collision slots is $G(n)L(n)P_{coll}$, the total number of read cycles in all binary tree recursive processes is $0.3922G(n)L(n)P_{coll}$. Let $R_{bits}$ be the string length of one read cycle and let $IDReq_{bits}$ be the string length of IDReq sent by the reader, respectively; $Bits_R$ can be finally expressed as follows:

$$Bits_R = 2(G_{bits} + L_{bits}) + \left[ G_{ini}L_{ini} + G(n)L(n) \right] QueryRep_{bits} + n \cdot IDTest_{bits} + L_{bits}.$$  (16)

4.3. Communication Complexity on Tag. Here we take $Bits_T$ for consideration to indicate the average amount of data transmitted by single tag. Because of the randomness of tag identification, the average bits of data sent by single tag are difficult to observe exactly, so the lower bound and the upper bound are introduced for reference.

(a) The Lower Bound. If one lets $lev(i)$ be the level of the ith leaf node in a strict binary tree with n leaf nodes, where i varies from 1 to n, according to the property of strict binary tree, the following equation can be easily derived:

$$\sum_{i=1}^{n} \frac{1}{2^{lev(i)}} = 1.$$  (17)

Since $\sum_{i=1}^{n} (1/2^{lev(i)}) = n \cdot (1/n) \sum_{i=1}^{n} (1/2^{lev(i)}) \geq n \cdot \sqrt{(1/2) \sum_{i=1}^{n} lev(i)}$, we can obtain

$$\sum_{i=1}^{n} lev(i) \geq n \log_2 n.$$  (18)

Expression (18) shows that the sum of levels of all leaf nodes in a strict binary tree with n leaf nodes is no less than $n \log_2 n$. Because the level of node indicates that the collision times tags allocated in it have experienced, and the colliding times in a strict binary tree with n tags become minimal only when the number of leaf nodes is equal to $n$, the total colliding times in a strict binary tree with n tags are no less than $n \log_2 n$. Let $P_{coll}$ and $P_{suc}$ denote the expected possibility of colliding slots and singleton slots, respectively; according to Assumption 3, the amount of data transmitted by tags in all colliding time slots satisfies

$$Bits_{CS} \geq G(n) \cdot L(n) \cdot P_{coll} \cdot (2.3922 \cdot \log_2 2.3922 + IDTest_{bits} + 2.3922 \cdot ID_{bits}).$$  (19)

The total amount of data sent by all tags, represented by $Bits_{TA}$, is composed of two parts: data sent by the tags in colliding time slots and data sent by the tags in singleton slots. The former is expressed by (19). Since every tag allocated in the singleton slot transmits IDTest as well as tag's ID only once, and there are $G(n)L(n)P_{suc}$ singleton slots with only 1 tag, $Bits_{TA}$ can be derived as the following expression:

$$Bits_{TA} = Bits_{CS} + G(n) \cdot L(n) \cdot P_{suc} \cdot (IDTest_{bits} + ID_{bits}) \geq G(n) \cdot L(n) \cdot \left( 2.3922 \cdot P_{coll} \cdot \log_2 2.3922 + P_{suc} \right) \cdot IDTest_{bits} + n \cdot ID_{bits}.$$  (20)
Finally, the lower bound of Bits\(_T\) can be obtained as follows:

\[
Bits_T \geq \frac{G(n) \cdot L(n)}{n} \cdot \left(2.3922 \cdot P_{\text{col}} \cdot \log_2 2.3922 + P_{\text{suc}}\right) \cdot \text{IDTestbits} + \text{IDbits}.
\]

(21)

(b) The Upper Bound. In order to measure the largest amount of data sent by tags, we must take the worst condition into account. Consider the following extreme situation. Let \(p_s\) be the probability of all colliding tags falling into the same child node simultaneously after splitting current collision node, and \(\lambda\) is the probability of only one tag falling into one child node. Obviously, \(p_s\) is much large than \(\lambda\) at the beginning of the binary tree recursive process, which means colliding tags will fall into the same child node after splitting. As the splitting process is ongoing recursively, \(p_s\) is decreasing and approaching \(\lambda\). If and only if \(p_s\) becomes not great than \(\lambda\) for the first time, the splitting of current collision node will result in one child node with only 1 tag and the other child node with the remainder tags, which separates 1 tag from the colliding tags and identifies it successfully. The splitting process will repeat recursively for the remaining tags in the colliding child node until there is no more than 1 tag falling into every node, or we can say the leaf node has been reached. Assuming that the \(i\)th identified tag has experienced \(k_i\) times of collision before being identified successfully and \(m\) denotes the total number of colliding tags in the binary tree, we can get

\[
\left[\left(\frac{1}{2}\right)^{m+1-i}\right]^{k_i} \leq \lambda,
\]

where \(k_i\) is an integer, \(k_i = \min(k_i)\)

\(\Rightarrow k_i = \left\lfloor \frac{1}{m+1-i} \log_{1/2} \lambda \right\rfloor\) \(\leq \frac{1}{m+1-i} \log_{1/2} \lambda + 1.
\]

So the amount of data sent by \(m\) tags in the binary tree can be considered as

\[
Bits_{CS} \leq \sum_{i=1}^{m} k_i \cdot \text{IDTestbits} + m \cdot \text{IDbits} \leq \left(\sum_{i=1}^{m-1} \left[ \frac{1}{m+1-i} \log_{1/2} \lambda \right] \right) \cdot \text{IDTestbits} + m \cdot \text{IDbits}.
\]

(23)

In the expression above, the fact that the range of \(i\) does not include \(m\) is because the \(m\)th tag will be identified undoubtedly in the next read frame if the \((m-1)\)th tag has been identified. Therefore, by expressions (22) and (23), we get the following expression:

\[
Bits_{CS} < \left\{ \left(\sum_{i=1}^{m-1} \left[ \frac{1}{m+1-i} \log_{1/2} \lambda \right] \right) \cdot \text{IDTestbits} + m \cdot \text{IDbits} + m \right\}.
\]

(24)

Seeing that \(\sum_{i=1}^{m}(1/i)\) is the sum of the first \(m\) terms of the harmonic series and its upper bound is \((\log_2 m + 1)\)\(^{[16]}\), expression (24) can be rewritten as

\[
Bits_{CS} < \left[ (\log_2 m + 1 - 1) \cdot \log_{1/2} \lambda + (m - 1) \right] \cdot \text{IDTestbits} + m \cdot \text{IDbits}.
\]

(25)

Considering \(m = 2.3922\), so the total amount of data sent by all tags is

\[
Bits_{TA} = Bits_{CS} + G(n) \cdot L(n) \cdot P_{\text{suc}} \cdot (\text{IDTestbits} + \text{IDbits}) \leq G(n) \cdot L(n) \cdot \left[ P_{\text{col}} \cdot \left( \log_2 2.3922 \cdot \log_{1/2} \lambda - 0.5820 \right) + P_{\text{suc}} \right] \cdot \text{IDTestbits} + n \cdot \text{IDbits}.
\]

(26)

And the average amount of data sent by single tag is shown as follows:

\[
Bits_T = \frac{Bits_{TA}}{n} < \frac{G(n) \cdot L(n)}{n} \cdot \left(\frac{P_{\text{col}} \cdot \left( \log_2 2.3922 \cdot \log_{1/2} \lambda - 0.5820 \right) + P_{\text{suc}}}{P_{\text{col}}} \right) \cdot \text{IDTestbits} + \text{IDbits}.
\]

(27)

Thus, the final expression of Bits\(_T\) can be deduced from expressions (21) and (27), which is expressed as an inequation with a lower bound and an upper bound as follows:

\[
Bits_T < \frac{G(n) \cdot L(n)}{n} \cdot \left( \left(2.3922 \cdot P_{\text{col}} \cdot \log_2 2.3922 + P_{\text{suc}}\right) \cdot \text{IDTestbits} + \text{IDbits} \right) - \left(2.3922 \cdot \log_2 2.3922 \cdot \log_{1/2} \lambda - 0.5820 \right) + P_{\text{suc}} \right) \cdot \text{IDTestbits} + \text{IDbits}.
\]

(28)

5. Simulation

Next, we need to verify how close the mathematic formulas presented in Section 4 match the reality. As the key factors influencing application for anticollision algorithms include the system throughput and the communication complexities, which indicate how fast tags can be identified (especially
important for mobile tags) and how low the energy is consumed, the comparisons for both of them between our algorithm and traditional ones should be made. So we program on Matlab7.1 and the simulations include (1) comparing theoretical values calculated by the mathematic formulas mentioned above with the simulative results acquired from simulations, (2) comparing the complexity of T-GDFSA with that of traditional algorithms including DFSA, EDFSA, BS, QT, and QTI, and (3) comparing the system throughput of T-GDFSA when the length of tag’s ID varies in a large range or the initial estimation of tag population becomes inaccurate. System throughput, denoted by S, is defined as the ratio of the number of tags to the sum of all time slots used in the read frames and all read cycles produced for querying the nodes of binary trees and can be obtained from the formula $S = nT_{\text{cycle}}/T$, where $T_{\text{cycle}}$ is the duration of one time slot as well as the duration of one read cycle.

Before simulation, some parameters need to be set properly. The parameters are set as follows: $p_1 = 1/L$, $L_{\text{bits}} = 14$, $G_{\text{bits}} = 5$, $\text{Lev}_{\text{bit}} = 4$, $R_{\text{bits}} = 23 (L_{\text{bits}} + G_{\text{bits}} + \text{Lev}_{\text{bit}})$, $\text{IDTest}_{\text{bits}} = 8$, $\text{IDReq}_{\text{bits}} = 8$, $\text{QueryRep}_{\text{bits}} = 4$, and $L_{\text{max}} = 256$. Considering that the possibility that each tag falls in a slot is $1/L$ when the frame size is $L$, we set $p_1 = 1/L$. $L_{\text{bits}}$ set to 14 means the maximal time slots in a read frame can be $2^{14}$. $G_{\text{bits}}$ set to 5 indicates that the maximal grouping size is $2^5$. $\text{Lev}_{\text{bit}}$ is set to 4, which refers to the fact that the maximal depth of the binary tree in recursive process is $2^4$. Obviously, $R_{\text{bits}}$ can be set to 23 as a sum of $L_{\text{bits}} + G_{\text{bits}} + \text{Lev}_{\text{bit}}$; $\text{IDTest}_{\text{bits}}$ and $\text{IDReq}_{\text{bits}}$ set to 8 means the reader and tags use a 1-byte string to achieve the strategy of slot test. The value of $\text{QueryRep}_{\text{bits}}$ is decided by the encoding method of commands and set to 4 in our simulations, which means each command sent by the reader is represented by a 4-bit binary string. As $L_{\text{max}}$, the maximal frame size, refers to the memory used in tags which is usually very limited in reality; it is just set to 256 in our simulations which means 1-byte memory is required in tags. Besides, the following parameters are tuned:

(i) $n$: the number of tags to be identified. It is increased by 50 in each trial in the range [50, 1000].
(ii) $L_{\text{ini}}$: the initial estimated number of unidentified tags. This value is used to set the initial read frame size. It is set to 0.5$n$, 1.5$n$, 2$n$, and 5$n$ for inaccurate estimation.
(iii) $\text{ID}_{\text{bits}}$: the length of tag’s ID. It is assigned to the value [48, 64, 96, 128] sequentially.
(iv) Running times: the times each simulation runs. It is set to be 100 times and the average result is taken for analysis.

In Figures 3–5, performance comparison of T-GDFSA between theoretical values and simulative results is shown. In Figure 3, the theoretical curve of $S$ appears as a zigzag line surrounding the curve of simulative results. The points of the wave crest of the theoretical curve correspond to the increase of grouping size, which reduces tags per group drastically and deteriorates $S$ significantly because of more extra idle slots. Figure 4 shows that the theoretical $\text{Bits}_{R}$ matches well the simulative one. In Figure 5, simulative values of $\text{Bits}_{S}$ are mostly 69 bits and the lower bound of theoretical values is near 58 bits, with the length of tag’s ID set to 48 bits. Notice that the upper bound of theoretical values is very close to simulative results when $\lambda$ is set to 0.01.

In Figure 6, compared with other traditional anticollision algorithms, T-GDFSA has the highest $S$ about 0.41 when tag’s ID is set to 48 bits long and uniformly distributed. Figure 7 shows that T-GDFSA has agreeable $\text{Bits}_{S}$ which is reduced by at least half of that of tree-based algorithms and nearly equal to that of ALOHA-based algorithms. In Figure 8, T-GDFSA has the lowest $\text{Bits}_{R}$ which means more power is saved in tags compared with other traditional algorithms.

Figure 9 shows the variation of $S$ with inaccurate initial tag estimation. When the estimated number is varying from 0.5$n$ to 1.5$n$, $S$ is still higher than 0.38, which indicates that
T-GDFSA has great tolerance for the inaccuracy of initial tag estimation. We also notice that T-GDFSA performs better in underestimation than overestimation. This is because smaller frame size in underestimation causes more tags to fall into colliding time slots, which can make full use of the advantages of T-GDFSA, while in the case of overestimation the identification process is more like an EDFSA identification, with more 1-tag time slots and idle time slots. Figure 10 shows that length variation of tag’s ID has little influence on $S$. When tag population exceeds 450, this effect can be approximately ignored.

The simulations above reflect the following. (1) Our mathematic formulas of the main metrics match well the reality. (2) Compared with traditional algorithms, T-GDFSA achieves higher system throughput while maintaining lower computing complexities, which is not available for traditional algorithms shown above. This significant advantage makes our algorithm easier to be realized in end-user applications.

6. Conclusion

In this paper, we firstly make a summary of traditional RFID anticollision algorithms, and then propose a novel hybrid
anticollision algorithm called T-GDFSA to improve the performance of RFID tag identification. T-GDFSA, which is based on grouped dynamic frame-slotted ALOHA and binary tree recursive process, has optimal group size and optimal read frame size and solves colliding time slots by introducing the binary tree recursive process. System model and mathematics formulas are present, and theoretical values of metrics show great agreement with the simulative results obtained from simulations. Compared with traditional ALOHA-based and tree-based anticollision algorithms, T-GDFSA exhibits excellent comprehensive performance with preferable system throughput around 0.41 and lower communication complexity which is similar to EDFSAbut at least half less than that of tree-based algorithms. Furthermore, T-GDFSA has tolerance for the inaccuracy of initial tag estimation and the length variation of tag’s ID.

As we can see, an accurate estimation of actual tag population in T-GDFSA is very significant. The estimation is used to set the optimal grouping size and frame size which ensures the performance. Meanwhile, the computing complexity of estimation cannot be very high to ensure the reader to identify tags promptly. However, the estimation in T-GDFSA follows TEM (i.e., Vogt’s method), which may not be optimal in both of time and accuracy, so one of the possible directions of our next work is to find a way to improve the estimation method for T-GDFSA to achieve better performance of RFID tag identification.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was sponsored by the National Natural Science Foundation of China (Grant nos. 61102034, 61172156, and 61071038), the Doctoral Fund of Guangdong University of Technology (Grant no. I13002), and Shenzhen Application Foundation key project (Grant no. JCYJ2013040109559824).

References

[1] S. S. Kim, Y. H. Kim, and K. S. Ahn, "An enhanced slotted binary tree algorithm with intelligent separation in RFID Systems," in Proceedings of the IEEE Symposium on Computers and Communications (ISCC ’09), pp. 237–242, Sousse, Tunisia, July 2009.

[2] X. Luo, Z. Xu, J. Yu, and X. Chen, “Building association link network for semantic link on web resources,” IEEE Transactions on Automation Science and Engineering, vol. 8, no. 3, pp. 482–494, 2011.

[3] J. M. Zhao, W. T. Li, and D.-A. Li, “Identifying the missing tags in categorized RFID systems,” International Journal of Distributed Sensor Networks, vol. 2014, Article ID 582951, 12 pages, 2014.

[4] G. Bagnato, G. Maselli, C. Petrioli, and C. Vicari, “Performance analysis of anti-collision protocols for RFID systems,” in Proceedings of the 69th Vehicular Technology Conference (VTC’09), pp. 1–5, IEEE, Barcelona, Spain, April 2009.

[5] W. Zhang, Y.-J. Guo, X.-M. Tang, G.-H. Cui, L.-K. Wu, and Y. Mei, “An efficient adaptive anticollision algorithm based on 4-ary pruning query tree,” International Journal of Distributed Sensor Networks, vol. 2013, Article ID 848746, 7 pages, 2013.

[6] X.-Q. Yan, Y. Liu, B. Li, and X.-M. Liu, “Numeric evaluation on the system efficiency of the EPC Gen-2 UHF RFID tag collision resolution protocol in error prone air interface,” International Journal of Distributed Sensor Networks, vol. 2014, Article ID 716232, 9 pages, 2014.

[7] V. Namboodiri, M. Desilva, K. Deegala, and S. Ramamoorthy, “An extensive study of slotted Aloha-based RFID anti-collision
[8] C. S. Keat and L. N. Shyan, "Dynamic framed slotted ALOHA algorithm for RFID systems with enhanced tag estimation technique," in Proceedings of the IEEE International Conference on RFID-Technologies and Applications (RFID-TA ’13), pp. 1–4, Johor Bahru, Malaysia, September 2013.

[9] J. R. Cha and J. H. Kim, “Dynamic framed slotted ALOHA algorithms using fast tag estimation method for RFID system,” in Proceedings of the 3rd IEEE Consumer Communications and Networking Conference (CCNC ’06), pp. 768–772, Las Vegas, Nev, USA, 2006.

[10] S.-R. Lee, S.-D. Joo, and C.-W. Lee, "An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification," in Proceedings of the 2nd International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous ’05), pp. 166–172, San Diego, Calif, USA, July 2005.

[11] D. R. Hush and C. Wood, "Analysis of tree algorithms for RFID arbitration," in Proceedings of the IEEE International Symposium on Information Theory (ISIT ’98), pp. 107–114, IEEE, Cambridge, Mass, USA, August 1998.

[12] C. Law, K. Lee, and K.-Y. Siu, "Efficient memoryless protocol for tag identification," in Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM ’00), pp. 75–84, Boston, Mass, USA, August 2000.

[13] J. Myung, W. Lee, J. Srivastava, and T. K. Shih, “Tag-splitting: adaptive collision arbitration protocols for RFID tag identification,” IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 6, pp. 763–775, 2007.

[14] H. Vogt, “Multiple object identification with passive RFID tags,” in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC ’02), pp. 6–9, Hammamet, Tunisia, 2002.

[15] M. A. Bonuccelli, F. Lonetti, and F. Martelli, “Instant collision resolution for tag identification in RFID networks,” Ad Hoc Networks, vol. 5, no. 8, pp. 1220–1232, 2007.

[16] Y. Langsam, M. J. Augenstein, and A. M. Tenenbaum, Data Structure Using C and C++ (2nd Edition), Prentice Hall, Englewood Cliffs, NJ, USA, 2nd edition, 1995.
