Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 674698, 9 pages
http://dx.doi.org/10.1155/2013/674698

Research Article

QSA: Query Splitting-Based Anticollision for Mobile RFID-Based Internet-of-Things

Jianliang Gao, Jianxin Wang, Jianbiao He, and Weiping Wang

School of Information Science and Engineering, Central South University, Changsha 410083, China

Correspondence should be addressed to Weiping Wang; wpwang@csu.edu.cn

Received 28 March 2013; Accepted 2 July 2013

Academic Editor: Lu Liu

Mobility is a common character of the emerging RFID-based internet-of-things. However, most of prior RFID anticollision algorithms ignore the movement of tags, which can degrade the identification performance seriously and even result in tag starvation problem. This paper presents a novel anticollision algorithm named Query Splitting-based Anticollision (QSA) for mobile RFID-based internet-of-things. By designing adaptive query, QSA reduces the number of collisions efficiently and makes it possible to identify multiple mobile tags without rollback. In QSA, we propose a query stack technology to avoid the rollback operation caused by new arriving tags, which solves the tag starvation problem under mobile environments. The performance evaluation shows that the proposed algorithm takes fewer timeslots and has better performance in identifying mobile tags.

1. Introduction

Radio frequency identification (RFID) is a contactless wireless communication technology. The scope for using this technology boosts as RFID tag becomes a low-cost device due to mass production [1]. As the rapid proliferation of RFID tags, it has given rise to various concepts that integrate the physical world with the virtual one. One of the most popular concepts is the Internet-of-Things (IoT), a vision in which the Internet extends into physical entities. In the IoT, RFID is the foundation to connect the things together [2].

RFID systems consist of a reading device called reader, and multiple tags which are attached to physical entities in the IoT. The reader is typically a powerful device with ample memory and computational resources. On the other hand, tags vary significantly in their computational capabilities. Among tag types, passive ones are emerging to be a popular choice for large scale deployments due to their low cost [3]. For passive tags, they respond only at reader commands with the energy provided by the reader [2, 4].

In RFID systems, the reader usually needs to communicate with multiple tags. If there are multiple tags in a reader's interrogation zone, the reader receives the responses from these tags simultaneously. For a reader, it is not able to distinguish exact information from the interfered wireless signals, which is called collision. Collision is a serious problem in RFID systems since the reader will not receive the messages rightly once collision occurs [5]. An example is shown in Figure 1. Simultaneous responses transmitted by multiple tags collide, resulting in an increase of identification delay, even failure of reading the tags. Therefore, an efficient anticollision algorithm is required to reduce collisions and to achieve fast identification.

The tag collision problem becomes more serious in mobile RFID-based IoT. The physical entities with tags are often mobile, which facilitates competitive advantage through benefits such as improved efficiency, increased visibility, reduced cost, and many others [6]. The mobile devices can be part of numerous products, gadgets, and vehicle parts [7]. These devices close the gap between the real word and its virtual representation via, for example, seamless identification and integration with other wirelessly-embedded devices and their surroundings.

However, the mobile devices result in new challenges for anticollision problem. One of the challenges is tag starvation. For example, as shown in Figure 1, the reader is processing the collision caused by Tag 1 to Tag N. Assuming the reader has partitioned the tags for several rounds and will identify one tag out right now. If Tag $N+1$ enters the reader's interrogation zone at this time, the new arrived tag might cause the rollback

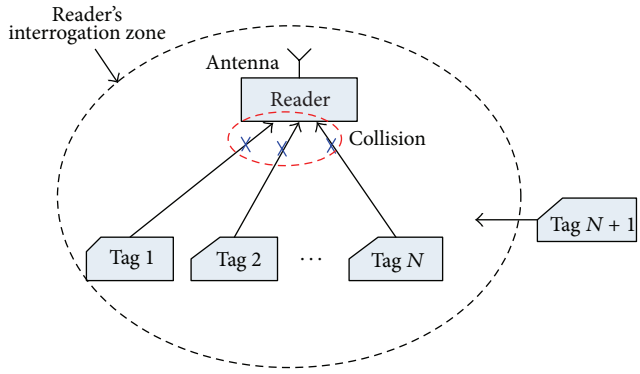

Figure 1: The tag collision with mobile tag.

of the partition operations. Therefore, the done operations have to be repeated and the delay of identifying the tags increases. If the new tags enter the reader's interrogation zone at an interval that causes rollback repeatedly, some tags might depart the reader's interrogation zone before being identified, which is called "tag starvation" in mobile environments. Therefore, it is necessary to provide efficient anticollision for mobile RFID-based IoT.

In this paper, we will deal with the "tag starvation" and performance problem for anticollision in mobile IoT. By the design of query stack and query rules, the proposed anticollision algorithm avoids repetition operations and decreases the collisions consequently. In summary, our contributions include the following.

(i) We characterize the problem of anticollision in mobile environment, which is more practical for the emerging IoT technology.

(ii) We propose a novel anticollision algorithm named Query Splitting-based Anticollision (QSA) for mobile RFID-based IoT, which solves the problem of tag starvation and performance degradation resulted from the tag mobility.

(iii) Simulation and analysis evaluate the efficiency of the proposed algorithm. The results show the better performance of the proposed algorithm to the prior methods.

The rest of this paper is organized as follows. Section 2 outlines preliminary, includes background, related work. Section 3 presents the problems for mobile tags. Section 4 gives the detailed design of our algorithm. The simulation results are provided in Section 5. Finally, Section 6 concludes the paper.

2. Preliminary

2.1. Collision Detection. Code technologies are widely used for collision detection. Manchester code is one of the most popular technologies for RFID systems [5]. In Manchester code, the value of one bit is defined as the voltage transition within a bit window. A bit "0" is coded by a positive transition, while a bit "1" is coded by a negative transition. In RFID systems, if two (or more) tags transmit different values simultaneously, the positive and negative transitions of the received bits cancel each other out. This state is not permissible in Manchester code during data transmission and is recognized as an error. Therefore, Manchester code makes it possible to "trace a collision to an individual bit" and "find where the collided bit is" [8].

Figure 2 shows an example of Manchester code for collision detection. The IDs of tag 1 and tag 2 are "10101100" and "10001001," respectively. When tags 1 and 2 send their IDs simultaneously using Manchester code, the decoded data from the interfered signal received by the reader is "10x01x0x," where "x" represents a collided bit. In this example, the locations of the collided bits are the 3rd, 6th, and 8th bits. This information helps the reader separate the collided tags into subsets more smartly and identify the tags more quickly.

Manchester code can be utilized to detect the collision bits, but the tags should be strictly synchronized. Fortunately, the tags in passive RFID systems are all driven by the reader with both energy and the same clock frequency.

2.2. Anticollision Algorithms. Many researches have focused on the issue of anticollision including tag-driven and reader driven procedures [9]. Tag-driven anticollision protocols function asynchronously [5]. For example, in Aloha-based protocols [10], time is divided into slots and each tag randomly transmits its ID in each timeslot. The tags continuously retransmit their IDs until the reader acknowledges their transmission. However, the Aloha-based protocols have several serious problems. For example, a specific tag may not be identified for a long time, leading to the so-called "tag starvation" problem. The performance of Aloha-based protocols is sensitive to the number of tags. Furthermore, it is very difficult to predict the number of tags in mobile environments, if not impossible.

For reader-driven anticollision protocols, they function synchronously, since all tags are controlled and checked by the reader simultaneously. Therefore, the reader can avoid tag starvation under static environments. Furthermore, they can be categorized into Binary Tree algorithm (BT) [11–13] and Query Tree algorithm (QT) [14–16].

2.2.1. Binary Tree Algorithm (BT). BT performs collision resolution by splitting collided tags into disjoint subsets. These subsets become increasingly smaller until they contain one tag. Each tag has a random binary number generator. For example, in Figure 3, tags with a counter value of zero are considered to be in the transmit state; otherwise, tags are in the sleep state. After each timeslot, the reader informs tags whether there is a collision or not. If there was a collision, each tag in the transmit state generates a random binary number. Tags will become in sleep state after being identified.

As can be seen that the average number of timeslots to identify the first tag is

$$T(N) = \log_2 N, \tag{1}$$

where N is the number of tags. Note that the reader needs to broadcast the universal condition (all bits are "1") before the

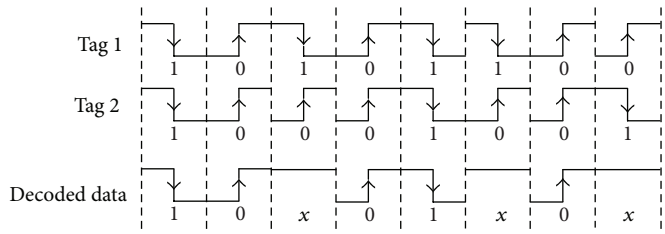

Figure 2: Collision detection with Manchester code.

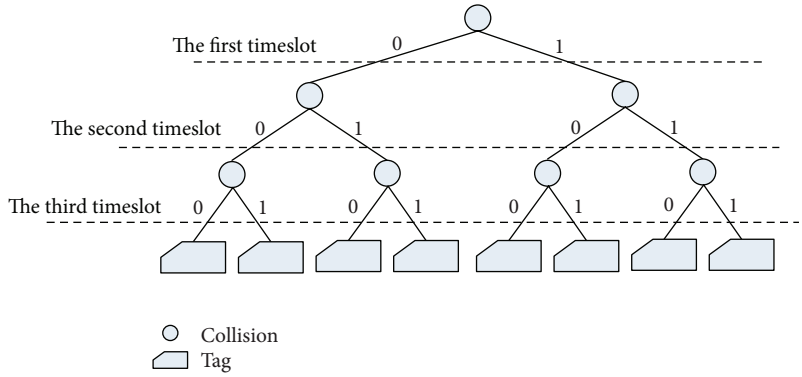

Figure 3: Binary tree based anticollision procedure.

first timeslots to collect all tags lying in its interrogation zero. Furthermore, the total number of timeslots for identifying all N tags is

$$T(N) = \sum_{k=1}^{N} \log_2 k. \tag{2}$$

The time consumption is the most serious defect since it has not recorded the history information, which results in a large amount of repeat operations.

2.2.2. Query Tree (QT). Query tree (QT) algorithms store tree construction at the reader, and tags only need to have a prefix matching circuit. The reader transmits a serial number to tags, which they then compare against their IDs. The tags whose IDs equal to or lower than the serial number respond to the reader's command. The reader then monitors tags reply bit by bit using Manchester code, and once a collision occurs, the reader splits tags into subsets based on collided bits. The reader then transmits another query by replacing the most significant collided bit with "0" and sets the other bits to "1." This procedure stops until a single tag has been selected out.

Rollback Query Tree (RQT) is proposed to reduce the average number of timeslots for identifying the tags. During the partition operation, these records are saved at the reader. Thus, the anticollision can avoid the repeating operations based on the saved information when the reader tries to select out the next tag. The timeslot number of RQT is

$$T(N) = \log_2 N + N - 1, \tag{3}$$

where N is the number of tags.

3. Problem for Identifying Mobile Tags

In static environments, Rollback Query Tree reduces the number of timeslots to deal with collision in RFID systems. However, both tag starvation and delay problems occurred in mobile environments. If one tag enters the reader's interrogation zone when the reader is processing the entered tags, some obtained results might be destroyed by the new arrived tag. The total number of identifying tags is

$$T(N + M) = T(N) + \sum_{i=1}^{M} (T_i + R_i), \tag{4}$$

where N, M are the numbers of the tags in the reader's interrogation zone already and the new arrived tags, respectively; $T(N)$ is the timeslots number as is defined in formula (3). T_i, R_i are the timeslots for processing the new arrived tag i and the timeslots that caused by the repeat operations. The increased timeslot for a new arrived tag i is $T_i + R_i$.

If the increased time is greater than the interval time between the arriving tags, the existing tags might not be processed in a long time since the reader has to reexecute the rollback operations for the new arrived tags. Therefore, tags starvation problem will happen if the following condition is met:

$$T_{i \to i+1} \leq T_i + R_i, \tag{5}$$

where $T_{i \to i+1}$ is the interval time between the new arrived tag i to $i + 1$. If each tag i meets the requirement in formula (5), no tags will be selected out successfully. This requirement is considerably strong, but to a moving tag, it will fail to be identified only if the reader has not processed before it leaves the reader's interrogation zone.

In the following, we take an example to illustrate the rollback caused by a new arriving tag. As shown in Figure 4, there are already four tags (IDs: 10100011, 11100010, 11100011, and 11110010) in the reader's interrogation zone. After two rounds of query (named q_1 and q_2), the tag 10100011 is the first one to be identified. Then, the reader should broadcast the queries q_3, q_4, and q_5 to select the next smallest tag 11100010 in the reader's interrogation zone. If a new tag 01100011 enters the interrogation zone between broadcasting q_4 and q_5, it will respond to the query q_5 "≤11100010." Then, the reader receives the response "$x110001x$" (combined by 11100010 and 01100011) instead of selecting out tag 11100010. Therefore, the reader should broadcast the new request command "≤11111111" according to the highest uncertain "x" bit. Unfortunately, this query command includes the responses from all tags, and the queries "q_3" and "q_4" have to be executed again in the following. In an extreme situation, the operation might be repeated many times for the consecutive arriving tags as shown in formula (5). Thus, tag starvation is possible for query tree schemes in mobile environments.

4. QSA: Query Splitting-Based Anticollision for Mobile IoT

4.1. Overview of the Anticollision Algorithm. Firstly, a set of queries $Q = (q_1, q_2, \ldots, q_l)$ is defined for the reader, where q_1 is initialized as "≤$\{1\}^n$". The reader executes the following steps to identify tags.

(1) Broadcast the current query in Q to all tags.

(2) When receiving the responses from tags:

(2.1) if the reply is string w without "x" bits, then select and process the tag with ID w;

(2.2) if a collision is detected; that is, the reply is string w with "x" bits, then set the next query in Q;

(2.3) if there is no reply from tags, do nothing.

(3) Update Q according to the received responses.

Repeat the above procedure until Q is empty. In this procedure, four commands (REQUEST, SELECT, READ-DATA, and DESELECT) are adopted as defined in Table 1.

For tags, let $w = w_1 w_2, \ldots, w_n$ be the tag's ID. The query is defined as follows: if $w \leq q$, then the tag sends string w to the reader, where q is the query string received from the reader.

As can be seen from the above procedure that the key problem is how to design the query scheme, our main idea is to keep the history records which can avoid rollback of the done operations.

4.2. Design of Adaptive Query

4.2.1. Query Stack Design for Rollback Operation. As shown in Table 1, REQUEST is the query command for the reader to split tags into subsets. To reduce the total timeslots for identifying all tags, we design a query stack structure to implement the rollback operations during query process. "Push Query" and "Pop Query" are the two basic operations of query stack. Each query response is pushed into query stack if and only if the response includes "x."

Table 1: Definition of commands.

REQUEST (ID_Condition)	Reader sends REQUEST command with parameter ID_Condition to tags. Tags compare their IDs against the received ID_Condition and reply their IDs to the reader if ID ≤ ID_Condition.
SELECT (ID)	Reader selects the tag ID.
READ-DATA	Reader reads data from the selected tag.
DESELECT	Reader cancels the selected tag, and this tag thus enters silent state.

We also take the example in Figure 4 to explain the stack operation. The response is "$111x001x$" for the query q_3: "≤11111111." The response "$111x001x$" includes "x"; thus, it is pushed into the query stack as shown in Figure 5. In the same way, the response "$1110001x$" is also pushed into the query stack with the query q_4: "≤11101111." When the reader broadcasts the query q_5: "≤11100010," a new tag "01100011" enters the reader's interrogation zone. Thus the response will be "$x110001x$" and it is also pushed into the stack as shown in Figure 5.

Once a response includes no "x," the reader will select this tag (using the command SELECT as shown in Table 1) and read the selected tag (using the command READ-DATA as shown in Table 1). After that, the reader will pop one element from the query stack. For example, when the new arrived tag "01100011" has been processed, the element "$x110001$" will then be popped out. The following design is how to set the "x" bits and broadcast new query commands.

4.2.2. Rule Design for Query Condition. The rule for query command is one of the most important designs in QSA. As shown in Table 2, the rules can be presented as two kinds. One is for the obtained response, and the other is for the pop stack.

(1) Rule for Obtained Response. When the reader receives response including "x" bits, the next query condition should be set according to the collision bits in the response. The second line in Table 2 shows the rule for this kind of obtained response.

In the first iteration, all bits are set as "1," that is, Max(ID), to collect the responses from all tags. For example, the parameter is "11111111" for eight bits ID. In the following iteration, the highest bit "x" is set as "0"; the bits which are lower than the highest "x" are all set as "1." According to the setting of "x" bits, we can draw the following theorem.

Theorem 1. *Rule for obtained response implements binary splitting for multiple tags.*

Proof. The response is a string $S = \{0, 1, x\}^n$, where n is the bit number of tag ID. The highest "x" bit in S is denoted as k, and it must meet the following requirement:

$$S[k] = \{x\}, \qquad S[n \cdots k+1] = \{0,1\}^{n-k}. \tag{6}$$

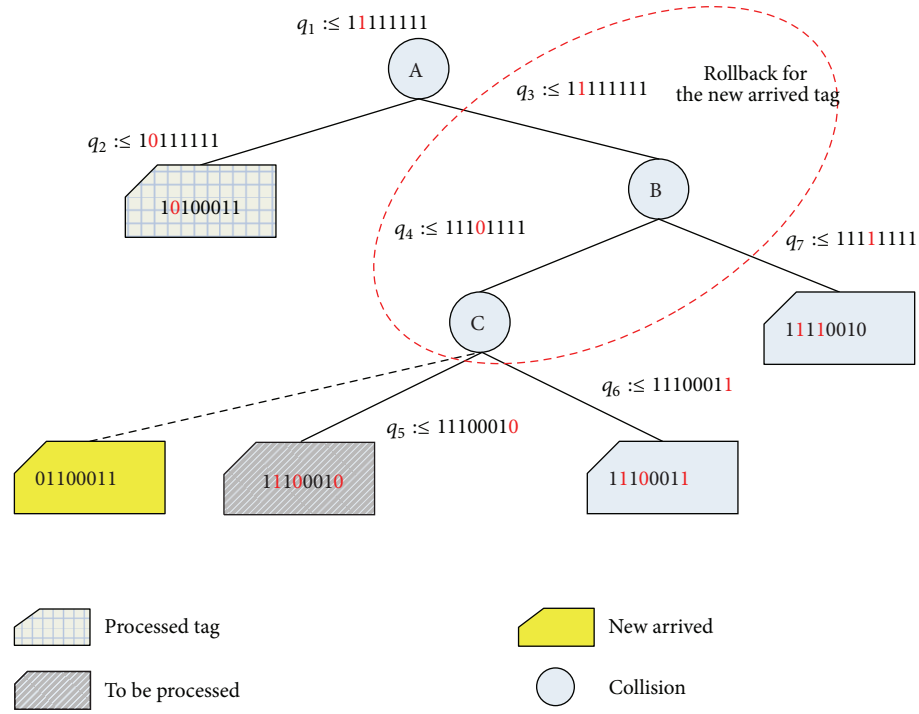

Figure 4: Rollback problem for the new arrived tag.

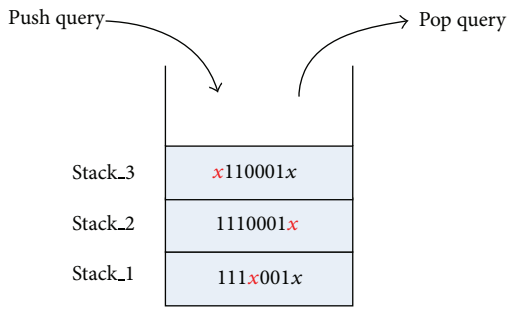

Figure 5: Query stack for new arrived tags.

Table 2: Rule design for query condition.

	The first iteration	The nth iteration
Rule for obtained response	Max(ID)	bit(k) = 0, bit($k-1\cdots1$) = $\{1\}^{k-1}$, (where k is the highest "x" bit in the response)

Rule for pop stack	Cannot happen	if there is new arrived tag
		bit(k) = 1;
		bit($k-1\cdots1$)= $\{0\}^{k-1}$
		else
		bit($k\cdots1$) = $\{1\}^{k}$
		end

Then, there is at least one tag whose kth bit is "1" and at least one tag whose kth bit is "0." Otherwise, the kth bit cannot be "0." Therefore, REQUEST "≤ bit(k) = 0, bit($k-1\cdots0$) = 1" splits the tags into two catalogs as binary division: one catalog is with $S[k] = \{0\}$, and the other is with $S[k] = \{1\}$. □

This rule can reduce the query scope by dividing the responding tags into two catalogs until only one tag respond.

(2) Rule for Pop Stack. After one tag is selected and processed, the top element is popped out. This element includes "x" bit. The rule for pop stack deals with the problem of setting these bits, which is shown in the third line of Table 2. Note that it cannot be the first iteration in anticollision process ("cannot happen" in Table 2) since the stack is empty at the beginning.

To the popped element, there are two situations: without new arrived tag and with arrived tag when it pushed into the stack. For the situation without new arrived tag, all bits are set as "1" in order to include all unprocessed tags; that is, bit($k\cdots1$) = $\{1\}^k$, where k is the highest "x" bit. On the other hand, if there are new arrived tags, the highest "x" bit is set as "1," and the lower bits are all set as "0"; that is, bit($k-1\cdots1$) = $\{0\}^{k-1}$.

Whether there is new arrived tag or not, it can be judged according to the following formula:

$$\text{Highest } x \ \text{(Top)} > \text{Highest } x \ \text{(Top-1)}, \tag{7}$$

where Highest x is the function to obtain the highest "x" bit in its parameter, Top means the first element in the query stack, and Top-1 is the second one.

Theorem 2. *If the highest "x" bit of the first stack element is higher than that of the second stack element, that is, the condition of formula* (7) *is met, there must be new arrived tag.*

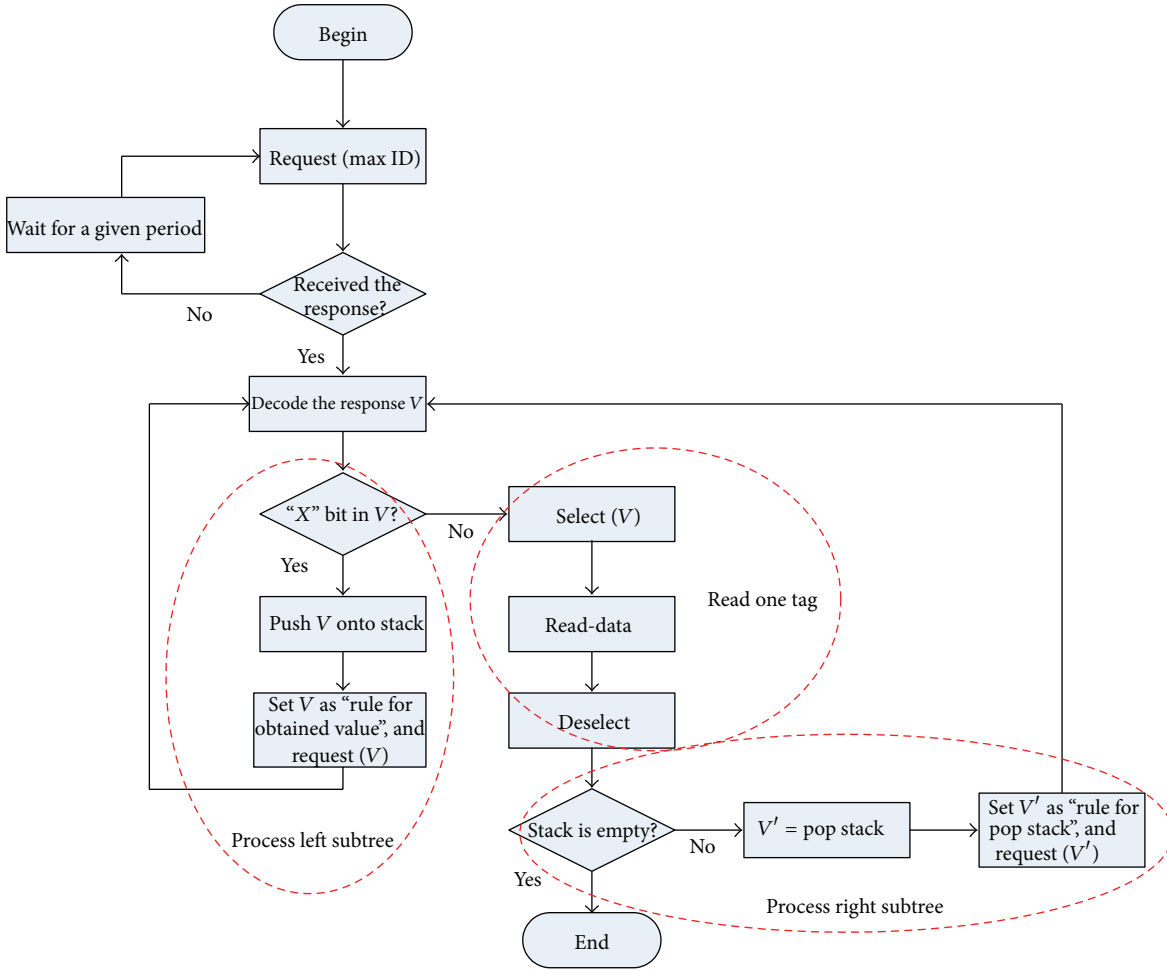

Figure 6: Flow chart of the proposed QSA algorithm.

Proof. According to rule for obtained response, the reader first processes the tags whose highest "x" bit is "0." Therefore, only when the higher "x" bit is processed, it is possible for lower "x" to be processed. Thus, the highest "x" always decreases as the distance to the top of query stack. However, it is possible that the "x" can be at any bit if new tags arrive. That is to say, formula (7) is a result caused by new arrived tags. □

For example, as shown in Figure 5, the highest "x" bit in the top stack element Stack_3 is 8, and that in Stack_2 is 1. Formula (7) meets and there is new arrived tag. In the example, the query for Stack_2 element "1110001x" is q_5 "≤11100010" in the example of Figure 4. While the highest "x" bit in top element Stack_1 is the 8th bit. There is new arrived tag whose 8th bit is "0." Otherwise, the new arrived tag is impossible to be included by the query "≤11100010" and the top element cannot be "x" at the 8th bit. In fact, there is a new arrived tag whose ID is 01100011 in the example.

4.3. The Procedure of the Proposed QSA. Based on the design of query stack and query rule, we describe the procedure of the proposed QSA algorithm in the following. In QSA, the reader first broadcasts the query with the parameter (max_ID) and receives the replies from all tags in its interrogation zone. If the received response of this query includes "x," the tree can be divided into the left subtree and the right subtree according to the highest bit "x." In the left subtree, the highest "x" bit is "0," and that of the right tree is "1." The procedure of the anticollision algorithm is shown in Figure 6. It mainly includes three parts: process the left subtree, process the right subtree, and read one tag.

4.3.1. Process the Left Subtree. If there is "x" in the received response, the reader begins to process the left subtree. Firstly, the reader pushes the response which has "x" bits into the query stack. We design this query stack which can record the entrance to the right subtree. Then, it sets the query condition according to the rule for obtained response in Table 2. Finally, the reader broadcasts the query with the set condition, which enables a new round communication.

4.3.2. Read One Tag. If the received response has no "x" bit, there is no collision and a single tag is identified. The reader will select this tag and read data from it. After the tag is

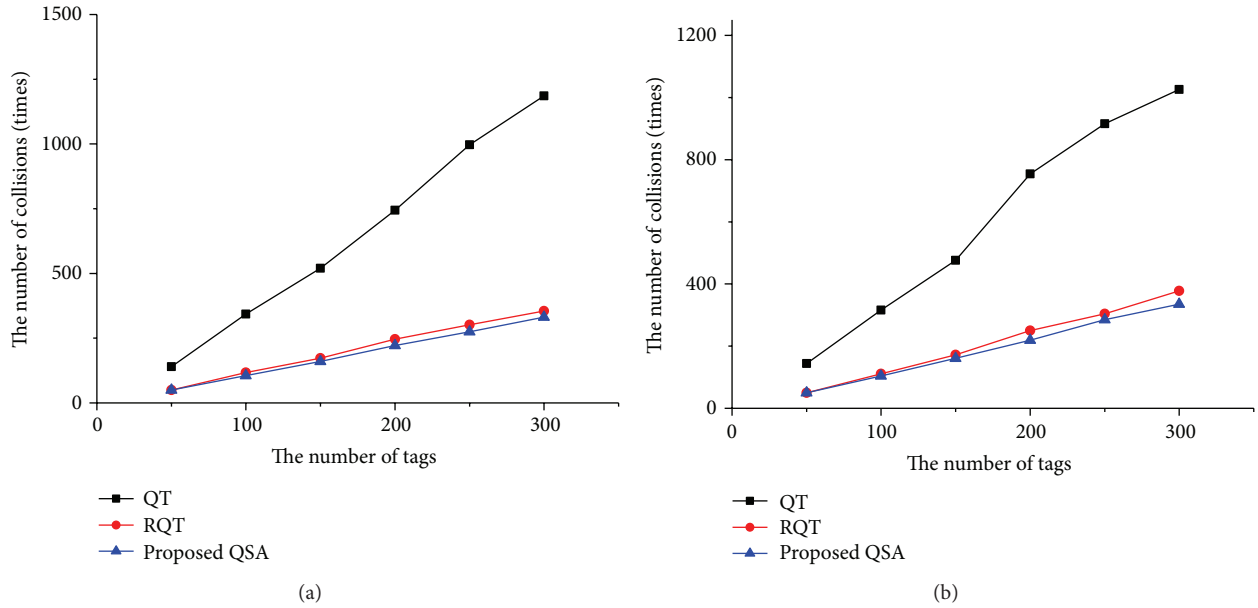

FIGURE 7: The number of collisions versus various numbers of tags. (a) Tag ID is 64 bits; (b) tag ID is 128 bits.

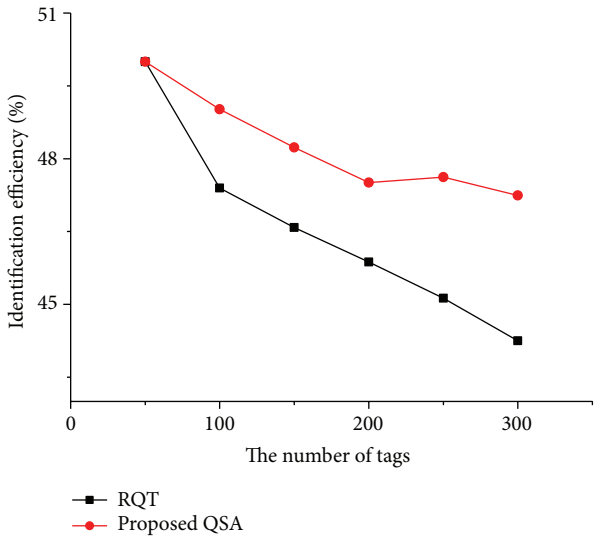

FIGURE 8: Identification efficiency.

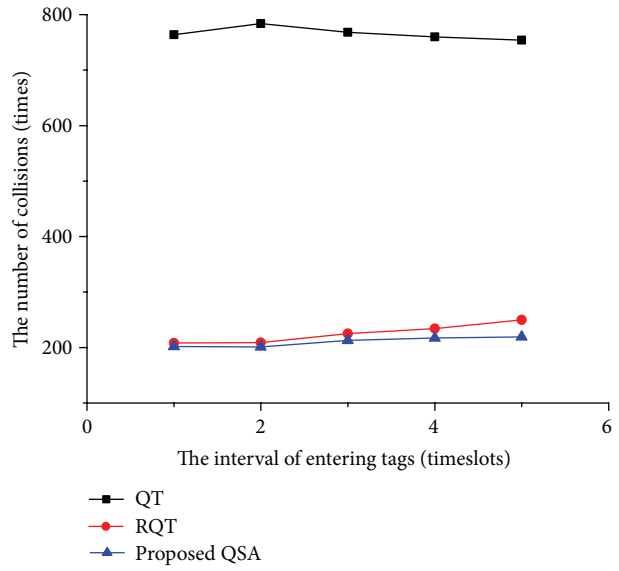

FIGURE 9: The number of collisions versus various mobile velocities.

processed, command "DESELECT" will make this tag keep silent state in the following.

4.3.3. Process the Right Subtree. After one tag is read, the algorithm will begin to process the right subtree. Firstly, the top element of the query stack is popped out, and it is set according to the rule for pop stack. As described in previous Section, there are two possibilities, that is, with new arrived tag and without new arrived tag when setting the query condition. If there is new tag which causes higher "x" bit, the query will enter the left tree process after receiving the response. Otherwise, the process will further deal with the right subtree. Note that the process will end until the query stack is empty, which means that all tags have been identified already.

5. Simulation Results

To evaluate anticollision performance, we compare the time consumption and identification efficiency of the proposed QSA algorithm with that of two representative schemes Query tree (QT) and Rollback Query Tree (RQT). Under mobile environments, we measure the number of timeslots used to read out all tags. Firstly, we fix mobile velocity of the tags and evaluate the results. The mobile velocity of tags is set

as the following rule: a group of five tags enter the reader's interrogation zero at the interval of five timeslots.

Figure 7 shows the results of collision numbers under various numbers of tags. As can be seen that QT meets the most number of collisions, since it has not recorded the query results during the procedure of finding the least tag ID. RQT always takes more collisions than the proposed QSA algorithm, which is consistent with formula (4). Both 64 bits and 128 bits tag ID in Figures 7(a) and 7(b), the proposed QSA takes the least number of collisions, which illustrates the efficiency of the QSA since it overcomes the rollback operations when new tags arrive.

To identify the tags as soon as possible, the algorithm should keep high identification efficiency. Identification efficiency is defined as the ratio of the number of success timeslots to the total timeslot number. Higher identification efficiency means less wasted timeslots for collisions. Figure 8 describes the comparison of identification efficiency between RQT and the proposed QSA. The identification efficiency of QSA is higher than that of RQT. It validates the analysis of number of consumed timeslots in Figure 7(a). The identification efficiency of the proposed QSA especially decreases slowly as the tag number increases. While the RQT is of lower identification efficiency when the tag number increases. For example, the identification efficiency of the RQT is below 45% when the tag number is 300.

From the theoretical analysis, it can be known that the mobile velocity will affect the results of collisions. Finally, we evaluate the impact of tag mobile velocity. In this experiment, total 200 tags enter the reader's interrogation zone at different intervals, which varies from 1 to 5 timeslots. The same with the above experiments, five tags come into the reader's interrogation zone at a pointed timeslot. As can be seen from Figure 9, the proposed QSA achieves the least number of collisions under all intervals of entering tags. An interesting change happens in QT scheme: the collision number increases firstly and then decreases. It is because the tag number increases quickly at the smaller interval of entering tags, which causes more collisions. The gap between RQT and the proposed QSA increases as the interval increases. It indicates that the velocity of mobile tags will cause different rollback operations in RQT, while it is eliminated in our QSA algorithm.

6. Conclusions

Collision is considered as one of the most important issues that affect the identification process in RFID systems. Mobile tags of IoT especially induce new problems such as performance degrade and tag starvation. In this paper, we propose a novel anticollision algorithm named QSA to solve these problems. The proposed QSA can overcome the problems resulted from mobile tags efficiently. Compared to prior schemes, the QSA presents a better performance at different mobile velocity and various tag numbers.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant nos. 61106036, 61173169, and 61272147, in part by the science and technology projects of Hunan province, and in part by State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences.

References

[1] A.-H. Mohsenian-Rad, V. Shah-Mansouri, V. W. S. Wong, and R. Schober, "Distributed channel selection and randomized interrogation algorithms for large-scale and dense RFID systems," *IEEE Transactions on Wireless Communications*, vol. 9, no. 4, pp. 1402–1413, 2010.

[2] F. Villanueva, V. David, and M. Francisco, "Internet of Things architecture for a RFID-based product tracking business model," in *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 811–816, 2012.

[3] Y.-H. Chen, S.-J. Horng, R.-S. Run et al., "A novel anti-collision algorithm in RFID systems for identifying passive tags," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 1, pp. 105–121, 2010.

[4] J. Panneerselvam, L. Liu, R. Hill, Y. Zhan, and W. Liu, "An investigation of the effect of cloud computing on network management," in *International Conference on High Performance Computing and Communications*, pp. 1794–1799, 2012.

[5] D. K. Klair, K.-W. Chin, and R. Raad, "A survey and tutorial of RFID anti-collision protocols," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 3, pp. 400–421, 2010.

[6] Y. Jiang and R. Zhang, "An adaptive combination query tree protocol for tag identification in RFID systems," *IEEE Communications Letters*, vol. 16, no. 8, pp. 1192–1194, 2012.

[7] S. Jiang, J. Miao, and L. Wang, "Mobile node authentication protocol for crossing cluster in heterogeneous wireless sensor network," in *Proceedings of the 3rd International Conference on Communication Software and Networks (ICCSN '11)*, pp. 205–209, May 2011.

[8] X. Jia, Q. Feng, and C. Ma, "An efficient anti-collision protocol for RFID tag identification," *IEEE Communications Letters*, vol. 14, no. 11, pp. 1014–1016, 2010.

[9] L. Kang, J. Zhang, K. Wu, D. Zhang, and L. Ni, "RCSMA: receiver-based carrier sense multiple access in UHF RFID systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 735–743, 2012.

[10] M. Al-Medhwahi, A. Alkholidi, and H. Hamam, "A new hybrid frame ALOHA and binary splitting algorithm for anti-collision in RFID systems," in *Proceedings of the 18th International Conference on Software, Telecommunications and Computer Networks (SoftCOM '10)*, pp. 219–224, September 2010.

[11] Y.-C. Lai and C.-C. Lin, "Two blocking algorithms on adaptive binary splitting: single and pair resolutions for RFID tag identification," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 962–975, 2009.

[12] Y.-C. Lai and C.-C. Lin, "A pair-resolution blocking algorithm on adaptive binary splitting for RFID tag identification," *IEEE Communications Letters*, vol. 12, no. 6, pp. 432–434, 2008.

[13] Y.-C. Lai and L.-Y. Hsiao, "General binary tree protocol for coping with the capture effect in RFID tag identification," *IEEE Communications Letters*, vol. 14, no. 3, pp. 208–210, 2010.

[14] Y. Lai, L. Hsiao, H. Chen, C. Lai, and J. Lin, "A novel query tree protocol with bit tracking in RFID tag identification," *IEEE Transactions on Mobile Computing*, pp. 1–13, 2012.

[15] J. H. Choi, D. Lee, and H. Lee, "Query tree-based reservation for efficient RFID tag anti-collision," *IEEE Communications Letters*, vol. 11, no. 1, pp. 85–87, 2007.

[16] J. Shin, B. Jeon, and D. Yang, "Multiple RFID tags identification with M-ary query tree scheme," *IEEE Communications Letters*, vol. 17, no. 3, pp. 604–607, 2013.