EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

A semi-blocking algorithm on adaptive query splitting for RFID tag identification

Yuan-Cheng Lai[1*], Chih-Chung Lin[2] and Chiung-Hon Leon Lee[3]

Abstract

Radio frequency identification (RFID) is a promising wireless technology for using tiny, remotely powered chips as identifiers. The number of RFID applications is rapidly increasing because RFID technology is convenient, fast, and contactless. However, collisions occur when multiple tags simultaneously transmit their IDs. Therefore, an efficient anti-collision algorithm is needed to accelerate tag identification. In some applications, the reader may repeatedly identify staying tags, which constantly exist in the reader's range. The adaptive query splitting algorithm (AQS) was proposed for reserving information obtained from the last identification process to enable rapid re-identification of staying tags. However, since AQS is a non-blocking algorithm that allows newly arriving tags to use the slots reserved for staying tags, collisions among them are problematic. Thus, *semi-blocking AQS* (*SBA*) proposed in this study is designed to reduce the collisions between arriving tags and staying tags by applying a semi-blocking technique in which only a minority of arriving tags use the slots reserved for staying tags. By counting the number of minor arriving tags, SBA estimates the number of unrecognized arriving tags and generates proper queries to minimize their collisions. The identification delay of SBA is analyzed, and simulation results show that SBA significantly outperforms AQS.

Keywords: RFID; Anti-collision; Tag identification; Blocking algorithm

1. Introduction

Radio frequency identification (RFID) has substantially replaced the conventional bar code system of automated identification because RFID is faster, more convenient, and contactless. The many RFID applications developed so far include inventory control, object tracking, and supply chain management. An RFID system usually contains a reader and multiple tags. Each tag has a unique identification (ID), and the reader recognizes all tags in its radio range through wireless communication. However, collisions occur when multiple tags transmit their IDs simultaneously because they share the same wireless channel. In this case, the reader cannot immediately recognize any tags, and the collided tags must retransmit their IDs until they are identified. This wastes bandwidth and delays identification. Therefore, an efficient anti-collision algorithm is needed to reduce tag collisions and to accelerate tag identification.

Current anti-collision algorithms can be classified as aloha-based algorithms and tree-based algorithms. Aloha-based algorithms estimate the number of tags and assign the proper number of slots to reduce the probability of tag collisions. They can be further classified as dynamic frame slotted aloha [1-5] and splitting frame slotted aloha [6-8]. The major difference is that the former dynamically adjusts the number of slots allocated for each frame and allows all unrecognized tags to share these slots while the latter splits unrecognized tags into different collision sets and allows each set to use individual slots separately. However, a limitation of aloha-based algorithms is the starvation problem, in which the tag cannot be identified because its responses are constantly colliding with others.

Tree-based protocols, which continuously split a set of tags into two subsets until all tags are identified, can be classified as query tree (QT) protocols [9-14] and binary tree (BT) protocols [15-18]. The former uses tag IDs while the latter adopts random binary numbers to split the set of collided tags. Therefore, QT is a memoryless protocol, i.e., the tag does not memorize information, whereas BT must maintain counters.

* Correspondence: laiyc@cs.ntust.edu.tw
[1]Department of Information Management, National Taiwan University of Science and Technology, No.43, Sec. 4, Keelung Rd., Taipei 106, Taiwan
Full list of author information is available at the end of the article

Many RFID applications require the reader to identify the same tags repeatedly. For example, the RFID applications for warehouse, parking lot, and library management, etc. In these applications, the objects, i.e., the goods, the cars, and the books, frequently stay at the same places for a long time. Therefore, the reader often identifies the same tags repeatedly. Therefore, if an anti-collision algorithm can reserve information obtained from the last process of tag identification, i.e., the last frame, the reader can skip many collisions and quickly re-identify the staying tags that remain within the range of the reader in the current frame.

Myung et al. [19-21] used two anti-collision algorithms, adaptive query splitting algorithm (AQS) and adaptive binary splitting algorithm (ABS), which were modifications of QT and BT, respectively, to retain information from the last frame. By using this information, AQS and ABS can successfully avoid collisions among staying tags. A previous study [21] showed that AQS requires lighter tag specifications although its performance may be affected by the distribution of tag IDs. Nevertheless, AQS and ABS are non-blocking algorithms in which newly arriving tags can use the slots reserved for staying tags. Thus, AQS and ABS cannot prevent arriving tags from colliding with staying tags, causing many collisions between them. Afterward, based on AQS and ABS, some blocking algorithms, which block arriving tags from using the slots reserved for staying tags, were proposed to prevent their mutual collisions [22-24]. However, these blocking algorithms need the information from the last frame to estimate the number of arriving tags.

Many RFID applications meet the situations that many arriving tags arrive sometimes and few arriving tags arrive sometimes, for example, people in the subway station or in the school, etc. In this case, the variation in the number of the arriving tags may be large. Therefore, when the blocking algorithms use the information from the last frame to do the estimation, their identification efficiency will be significantly reduced.

The novel *semi-blocking AQS* (*SBA*) algorithm proposed in this paper inherits the essence of AQS by using staying tag information obtained from the last frame. By applying a semi-blocking technique that allows a minority of arriving tags to use the slots reserved for staying tags, SBA reduces collisions between arriving tags and staying tags. Moreover, by counting the number of minor arriving tags, SBA can estimate the number of unrecognized arriving tags and generates proper queries to minimize their mutual collisions.

This study makes the following contributions to the literature. (1) We propose a semi-blocking algorithm, SBA, to quickly identify staying tags and arriving tags. SBA estimates the number of arriving tags according to the information obtained in the current frame rather than in the last frame. (2) We mathematically analyze the identification delay, i.e., the number of total required slots, of SBA. (3) We conduct extensive simulations and investigate the performance of SBA. The simulation results show that SBA significantly outperforms AQS.

The rest of this paper is organized as follows. Section 2 briefly describes QT, AQS, and other related works. Section 3 presents the underlying concept and operation of SBA. In Section 4, the mathematical analysis of identification delay for SBA is derived. Section 5 discusses the simulation results confirming the superiority of SBA over AQS. The conclusions are presented in Section 6.

2. Related works

Some terms are first defined as follows.

- *Frame*: a frame is the duration from the moment a reader begins recognizing tags within its reading range to the time it completes all recognition. Let f_i denote the i-th frame.
- *Slot*: a slot is the duration that a reader transmits a query signal to the tags and then the tags respond by sending their IDs to the reader. Depending on the number of tag responses, a slot is called *idle*, *readable*, or *collision* when no tag responds, one tag responds, or multiple tags respond, respectively.
- *Staying tag in the i-th frame*: the tag exists in f_{i-1} and also in f_i.
- *Arriving tag in the i-th frame*: the tag does not exist in f_{i-1} but appears in f_i.
- *Leaving tag in the i-th frame*: the tag exists in f_{i-1} but disappears in f_i.
- *Possible tags in the i-th frame*: the tags appear and are recognized in f_{i-1} and are likely to appear in f_i. Possible tags are the combination of staying tags and leaving tags.

2.1. QT

The QT splits a tag set by tag IDs [9-14]. The reader owns a queue Q, which stores bit strings of the queries and is initialized with two 1-bit strings, 0 and 1, at the beginning of each frame. At each slot, the reader interrogates the tags by popping one string from Q and transmitting it to the tags. If the prefix of a tag ID matches the bit string of the query, the tag responds by transmitting its ID. The reader can identify the tag when only one tag responds with its ID. When the tags' responses mutually collide, the query $q_1q_2...q_x$ is called a collision query. To solve this collision set composed of multiple tags, the reader pushes two 1-bit longer queries, $q_1q_2...q_x0$ and $q_1q_2...q_x1$ into Q. Thus, the set of tags with prefix $q_1q_2...q_x$ is then split into two subsets of tags, one with prefix $q_1q_2...q_x0$ and another with prefix $q_1q_2...q_x1$.

The two subsets will respond with their IDs at separate slots according to the queries sent from the reader. The reader continues expanding the query until one or no response is received. When Q is empty, the reader concludes that all of the tags have been recognized.

2.2. AQS

The AQS, which was modified from QT, preserves readable queries obtained from the last frame in order to avoid unnecessary slots generated from identifying staying tags [20,21]. It also retains idle queries acquired from the last frame in order to interrogate arriving tags during the current frame. The reader in the AQS has two queues: Q, which has the same manipulation as QT, and CQ, a candidate queue that gathers readable queries and idle queries in the last frame and retains them until the current frame starts. At the beginning of each frame, the reader first checks whether CQ has any query. An empty CQ means that there is no tag recognized in the last frame, making the reader initialize Q with two 1-bit strings, 0 and 1. On the other hand, if CQ has some queries, the reader initializes Q by copying all queries from CQ. Then, the reader starts the identification procedure as QT until Q is exhausted.

Also, during the identification procedure, whenever a readable query or an idle query occurs, the reader stores this query into CQ. Thus, in the next frame, the collisions among the staying tags can be totally avoided. However, leaving tags generate some unnecessary idle queries. AQS uses a query deletion procedure to handle this situation. For a node of a collision query, abnormal queries are its two child nodes being a pair of node types as follows: (1) a readable query and an idle query and (2) two idle queries. If leaving tags transform a pair of child nodes as abnormal queries, the reader then deletes unnecessary queries. That is, when $q_1q_2..q_x0$ and $q_1q_2..q_x1$ are abnormal queries, the reader deletes $q_1q_2..q_x0$ and $q_1q_2..q_x1$ from CQ and enqueues $q_1q_2..q_x$ into CQ. The reader deletes all abnormal queries from CQ recursively until all the nodes do not have a child node pair being abnormal queries.

2.3. Other works

The ABS, which is based on BT, has the similar concept to AQS [19,21]. The ABS may have a shorter identification delay compared to AQS, in which performance depends on the distribution of tag IDs [21]. In each tag, however, AQS only requires a matcher while ABS must maintain two counters. Therefore, AQS has lighter tag requirements compared to ABS.

Since both AQS and ABS are non-blocking algorithms, they cannot prevent arriving tags from colliding with staying tags, resulting in numerous collisions. Therefore, two protocols, the single resolution blocking ABS algorithm (SRB) and the pair resolution blocking ABS algorithm (PRB) [22,23], were proposed to address these issues. The blocking technique performed by SRB and PRB enables the use of different slots by arriving tags and staying tags, which prevents the former from colliding with the latter. However, since they are based on ABS, they require higher specifications in tags. Afterward, based on AQS, the other two blocking protocols, the couple-resolution blocking algorithm (CRB) and the enhanced couple-resolution blocking algorithm (ECRB) [24], were proposed. CRB and ECRB actually need lower tag specifications than SRB and PRB. However, CRB and ECRB estimate the number of arriving tags in the current frame based on the information obtained from the last frame. When the number of arriving tags significantly changed, these blocking algorithms, including SRB, PRB, CRB, and ECRB, cannot accurately estimate the number of arriving tags, resulting in more collisions when identifying them.

Efficient continuous scanning (ECS) based on slotted aloha also identifies staying tags quickly by the information gathered in the last frame [25]. The ECS is composed of two phases: the first phase identifies arriving tags while the second phase finds the leaving tags. In the first phase, arriving tags and staying tags individually select a slot in a round. Only arriving tags selecting pre-empty slots, i.e., no staying tag responds in the slots, are allowed to be active and will be identified in the next round using typical slotted aloha. In the second phase, if a pre-single slot, i.e., only one possible tag selects in the slot, is empty, the corresponding possible tag leaves. Clearly, collisions between arriving tags and staying tags still occur since ECS is a non-blocking algorithm. ECS relieves this problem by the use of different slot durations, so it can obtain acceptable performance. However, the drawback is that ECS cannot provide perfect accuracy. Some arriving tags may not be identified and some leaving tags may not be detected either.

Compared to previous approaches such as AQS, ABS, SRB, PRB, CRB, ECRB, and ECS, the proposed SBA is superior in five ways. First, SBA is a semi-blocking algorithm whereas other algorithms are either non-blocking or blocking algorithms. Thus, SBA is a more generic approach. Second, SBA is based on QT whereas SRB, PRB, and ECS are based on either slotted aloha or BT. Thus, SBA requires only light tag specifications. Third, SBA can estimate more accurate number of arriving tags because it uses the information obtained in the current frame while others use the information gathered in the last frame. Fourth, PRB, CRB, and ECRB use a pair resolution technique which couples possible tags and thus need less time for identifying staying tags. However, when the wireless channels are error-prone, these algorithms will generate false-positive results [23]. On the other hand, SBA does not have this problem. Fifth, SBA

provides perfect accuracy on identification while ECS may fail to identify some arriving tags and may fail to detect some leaving tags. Table 1 summarizes the differences between these algorithms.

3. Semi-blocking AQS

Although AQS can avoid collisions among staying tags, it has two drawbacks. First, it cannot prevent arriving tags from colliding with staying tags. Second, it reserves idle queries obtained from the last frame in order to interrogate arriving tags whose ID prefix matches one of these queries, but these queries may again cause idle slots in the current frame when no such arriving tag appears. The proposed SBA algorithm avoids these problems. The AQS and SBA differ in three ways. First, the former is a non-blocking algorithm while the latter is a semi-blocking algorithm to reduce the collisions between staying tags and arriving tags. Second, SBA does not retain the idle queries of the last frame. Third, SBA estimates the number of arriving tags so that the appropriate queries can be generated to identify them.

The SBA is performed in two phases. The first phase identifies staying tags and a few arriving tags. In the phase, since SBA blocks most arriving tags, the probability of a staying tag colliding with arriving tags is low. The second phase identifies arriving tags that are not recognized in the first phase. To avoid excessive collisions caused by arriving tags, SBA counts the number of minor arriving tags that have responded in the first phase to estimate the number of arriving tags that will respond in the second phase. By generating proper queries, SBA significantly reduces collisions caused by these unrecognized arriving tags.

3.1. Estimating the number of arriving tags

The key issue in SBA is accurately estimating the number of arriving tags that will respond in the second phase. The following notations are used to describe this estimation method.

P_1: the probability that an arriving tag in which the ID prefix matches the query is allowed to respond in the first phase.

N_1: the exact number of arriving tags responding in the first phase.

$\hat{N}$: the estimated number of total arriving tags.

$\hat{N}_2$: the estimated number of arriving tags responding in the second phase.

Q: a set $[q_1, q_2, q_3, \ldots q_m]$ where q_i denotes the i-th readable query and m is the total number of readable queries in Q.

Obviously, $\hat{N}_2 = \hat{N} - N_1$. An intuitive thought to derive $\hat{N}$ is $\hat{N} = N_1/P_1$. However, this is not correct. In SBA, Q only stores the readable leaf nodes in the query tree of the last frame and does not store the idle leaf nodes. Thus, the arriving tags located in the idle leaf nodes have no chance to respond since they are not interrogated by the reader in the first phase. For arriving tags that are involved in the first phase, only the arriving tags located in the readable nodes can be interrogated. Therefore, the percentage P_r of arriving tags interrogated in the first phase is calculated as

$$P_r = \sum_{i=1}^{m} \frac{1}{2^{|q_i|}}, \tag{1}$$

where $|q_i|$ is the length of q_i. Obtaining P_r enables

$$\hat{N} = \left\lceil N_1/(P_1 \times P_r) \right\rceil, \tag{2}$$

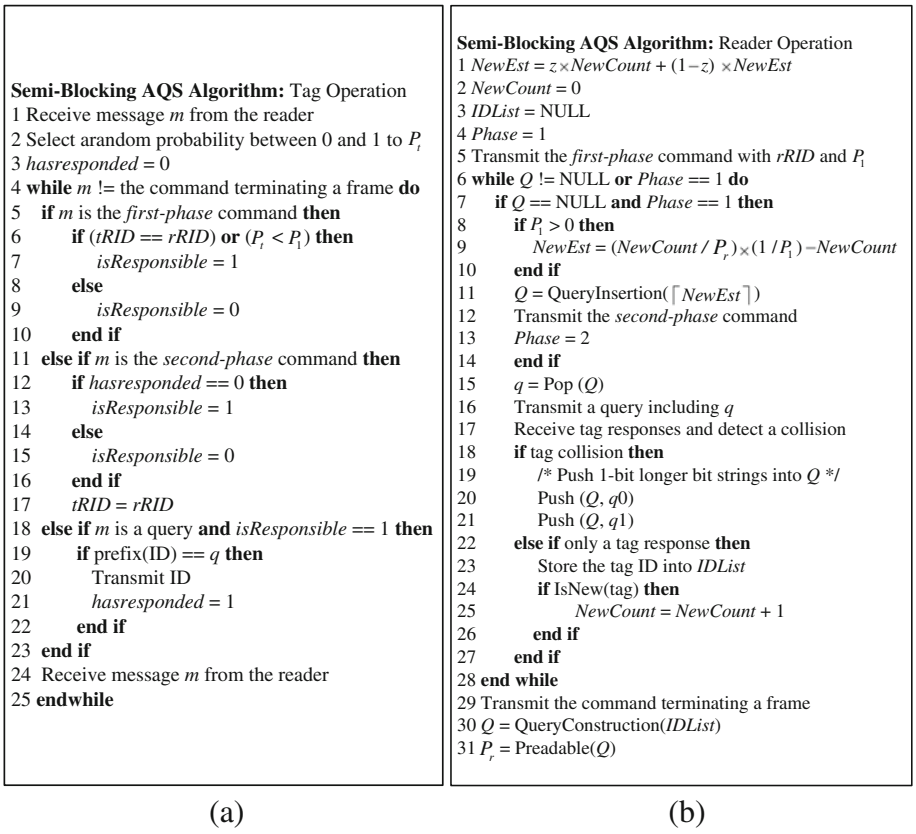
$$\hat{N}_2 = \left\lceil N_1/(P_1 \times P_r) - N_1 \right\rceil. \tag{3}$$

3.2. The SBA procedure

The unique ID of the SBA reader is *rRID*. Since each tag stores its associated reader ID, *tRID*, a tag can independently determine whether it is a staying tag or an arriving tag. For the SBA pseudocode in Figure 1, panels a and b are the tag and the reader operations, respectively. Like AQS, the reader in SBA has a queue, Q, but it only

Table 1 Comparison among different anti-collision algorithms

Algorithm	Blocking/non-blocking	Type	Information for estimation	Pairing resolution	Correct identification
AQS	Non-blocking	QT-based	Last frame	No	Yes
ABS	Non-blocking	BT-based	Last frame	No	Yes
SRB	Blocking	BT-based	Last frame	No	Yes
PRB	Blocking	BT-based	Last frame	Yes	No in error-prone channel
CRB	Blocking	QT-based	Last frame	Yes	No in error-prone channel
ECRB	Blocking	QT-based	Last frame	Yes	No in error-prone channel
ECS	Non-blocking	Aloha-based	Last frame	No	No
SBA (this work)	Semi-blocking	QT-based	Current frame	No	Yes



Semi-Blocking AQS Algorithm: Tag Operation

```
1 Receive message m from the reader
2 Select arandom probability between 0 and 1 to P_t
3 hasresponded = 0
4 while m != the command terminating a frame do
5   if m is the first-phase command then
6       if (tRID == rRID) or (P_t < P_1) then
7           isResponsible = 1
8       else
9           isResponsible = 0
10      end if
11  else if m is the second-phase command then
12      if hasresponded == 0 then
13          isResponsible = 1
14      else
15          isResponsible = 0
16      end if
17      tRID = rRID
18  else if m is a query and isResponsible == 1 then
19      if prefix(ID) == q then
20          Transmit ID
21          hasresponded = 1
22      end if
23  end if
24  Receive message m from the reader
25 endwhile
```

(a)

Semi-Blocking AQS Algorithm: Reader Operation

```
1  NewEst = z×NewCount + (1−z) ×NewEst
2  NewCount = 0
3  IDList = NULL
4  Phase = 1
5  Transmit the first-phase command with rRID and P_1
6  while Q != NULL or Phase == 1 do
7      if Q == NULL and Phase == 1 then
8          if P_1 > 0 then
9              NewEst = (NewCount / P_r)×(1 / P_1) −NewCount
10         end if
11         Q = QueryInsertion(⌈NewEst⌉)
12         Transmit the second-phase command
13         Phase = 2
14         end if
15         q = Pop (Q)
16         Transmit a query including q
17         Receive tag responses and detect a collision
18         if tag collision then
19             /* Push 1-bit longer bit strings into Q */
20             Push (Q, q0)
21             Push (Q, q1)
22         else if only a tag response then
23             Store the tag ID into IDList
24             if IsNew(tag) then
25                 NewCount = NewCount + 1
26             end if
27         end if
28 end while
29 Transmit the command terminating a frame
30 Q = QueryConstruction(IDList)
31 P_r = Preadable(Q)
```

(b)

Figure 1 Pseudocode of the SBA algorithm. (a) Tag operation. **(b)** Reader operation.

reserves readable queries obtained from the last frame. At the start of a frame, the reader first transmits the *first-phase* command with *rRID* and P_1 to all tags (line 5, Figure 1b). Upon receiving the *first-phase* command with *rRID* and P_1 from the reader, each tag checks whether its *tRID* matches the *rRID* sent by the reader. If so, the tag interprets itself as a staying tag and directly changes its variable, *isResponsible*, to 1. However, if *tRID* does not equal *rRID*, the tag interprets itself as an arriving tag. Then, each arriving tag generates its own random probability P_t for determining whether to respond to the query in the first phase or in the second phase (lines 6 to 10, Figure 1a). When P_t is smaller than P_1, the arriving tag can respond to the query in the first phase, so it changes its *isResponsible* to 1. Otherwise, it responds in the second phase and thus sets its *isResponsible* to 0. Afterward, the reader sends queries acquired from Q, which stores the readable queries obtained from the last frame, until Q is empty.

In the first phase, the reader and the tags that can respond in the first phase, including all staying tags and minor arriving tags, operate as QT. Thus, if some arriving tags collide with staying tags in the first phase, the SBA reader pushes two 1-bit longer queries into Q to solve these collisions. Moreover, in the first phase, the reader also stores each identified tag ID in *IDList* (line 23, Figure 1b) and counts the number of arriving tags as *NewCount*. Each tag also sets *hasreponded* to 1 if it has responded (lines 19 to 22, Figure 1a).

At the end of the first phase, i.e., when Q is empty and *Phase* equals 1, the reader has identified all staying tags and minor arriving tags that can respond in the first phase. Before starting the second phase, the reader uses Equation 3 to predict the number of unrecognized arriving tags, *NewEst*, by *NewCount*, which records the number of recognized arriving tags (line 9, Figure 1b). After obtaining *NewEst*, the QueryInsertion() function operated internally in the reader generates a complete binary tree in which the number of leaf nodes equals $\lceil NewEst \rceil$ and returns all leaf nodes into Q (line 11, Figure 1b). Based on the characteristics of a complete binary tree, its leaf nodes can cover all possible ID prefixes. Thus, the reader can use these queries of leaf nodes to interrogate arriving tags in the second phase.

The QueryInsertion() function operates as follows. Let $2^{l-1} < \lceil NewEst \rceil \le 2^l$ and the numbers of $(l-1)$-bit and

l-bit queries be denoted as c_1 and c_2, respectively. Then, two simultaneous equations can be easily obtained as

$$\begin{cases} c_1 + c_2 = \lceil NewEst \rceil \\ 2c_1 + c_2 = 2^l \end{cases}$$

The second formula is because c_1 queries of $(l - 1)$ bits and c_2 queries of l bits must construct a complete prefix set of all IDs. By solving the simultaneous equations, c_1 is $2^l - \lceil NewEst \rceil$ and c_2 is $2\lceil NewEst \rceil - 2^l$. Then, c_2 queries of l bits are first generated and c_1 queries of $l - 1$ bits are then generated according to the increasing order of binary representations of these queries. For example, when $\lceil NewEst \rceil = 5$, the QueryInsertion() function generates two 3-bit queries and three 2-bit queries, i.e., 000, 001, 01, 10, and 11.

Upon receiving the *second-phase* command, each recognized tag changes its *isResponsible* to 0 while other unrecognized tags change their *isResponsible* to 1 (lines 11 to 16, Figure 1a). Also, in the second phase, each tag that can respond in the second phase and the reader are operated as QT. The reader terminates the identification process when Q is empty.

Unlike AQS, SBA cannot immediately store readable queries obtained during the identification process because the readable queries obtained from recognizing staying tags and from recognizing arriving tags may overlap. For example, SBA may obtain readable query 00 for staying tag 0001 and also obtain readable query 00 for arriving tag 0010. To find all readable queries of a frame, the reader applies the QueryConstruction() function to *IDList* (line 30, Figure 1b). This function uses all recognized tag IDs stored in *IDList* to internally operate the QT algorithm and find correct readable queries (000 and 001 in the example above). The SBA also uses the Preadable() function to determine probability P_r based on Equation 1 (line 31, Figure 1b).

3.3 Special case of SBA: a blocking algorithm

A special case of SBA is that no any arriving tag can be allowed to use the slots reserved for staying tags, i.e., $P_1 = 0$. In this case, SBA becomes a blocking algorithm that completely prevents arriving tags from colliding with staying tags. However, since no information can be obtained from the first phase, SBA cannot use Equation 3 to estimate the number of arriving tags. Therefore, SBA changes the method to adopt the information from the last frame to estimate the number of arriving tags. Specifically, the reader estimates the number of arriving tags, *NewEst*, in the current frame with an exponential average of *NewCount*, which records the number of total arriving tags in the last frame, i.e., $NewEst = z \times NewCount + (1–z) \times NewEst$ (line 1, Figure 1b). Factor z is used to weight the last estimation and the exact number of arriving tags in the last frame.

3.4. Hardware cost and energy consumption

The SBA tag needs a random number generator and properly maintains four variables, P_t, *tRID*, *isResponsible* (1 bit), and *hasreponded* (1 bit). On the other hand, the AQS tag requires only one matcher to match the prefix and its ID. Thus, the cost of SBA tags may increase. However, as defined in ISO 18000–6 [26], a tag shall have a random/pseudorandom number generator and the minimal memory size of 4 bytes. Thus, the hardware requirements for SBA can be supported for existing tags. Moreover, considering the energy consumption caused by communication, SBA consumes less energy than AQS because the former spends less slots and transmits less bits than the latter, as shown in Section 5.

4. Performance analysis

In this section, we analyze the identification delay of SBA in the average case. For convenience, the number of total slots required in QT is first derived because SBA uses QT upon encountering a collision slot. Let T_i represent the set of tags existing in f_i, and let n be the number of tags in the set T_i, i.e., $n = |T_i|$. Additionally, let α and β be the numbers of arriving tags and leaving tags, respectively, in f_{i+1}. Finally, assume that the bit length b of each tag ID is infinite and that tag IDs are uniformly distributed among 2^b IDs. Thus, the reader can continually extend the prefix by appending bits until all tags are recognized. Since the size of b is always sufficient, this assumption appears reasonable and does not substantially affect the accuracy of the following analyses.

4.1. QT

In QT, slots in a frame can be represented as a query tree. Thus, the average number of total collision slots for n tags, $C_{QT}(n)$, is as follows:

$$C_{QT}(n) = \sum_{k=1}^{\infty} C_{QT}(n,k), \tag{4}$$

where $C_{QT}(n, k)$ is the average number of collision slots in the depth k of the query tree for n tags. Since the total number of nodes in the depth k of a full query tree is 2^k, $C_{QT}(n, k)$ can be written as

$$C_{QT}(n,k) = 2^k\left(1–PI_{QT}(n,k)–PR_{QT}(n,k)\right), \tag{5}$$

where $PI_{QT}(n, k)$ and $PR_{QT}(n, k)$ are the probabilities of a slot being an idle slot and a readable slot in the depth k

of the query tree, respectively. These probabilities are easily calculated as

$$PI_{QT}(n,k) = \left(1-\frac{1}{2^k}\right)^n, \qquad (6)$$
$$PR_{QT}(n,k) = n\left(\frac{1}{2^k}\right)\left(1-\frac{1}{2^k}\right)^{n-1}.$$

From Equations 5 and 6, we obtain

$$C_{QT}(n,k) = 2^k\left\{1-\left(1-\frac{1}{2^k}\right)^n - n\frac{1}{2^k}\left(1-\frac{1}{2^k}\right)^{n-1}\right\}. \qquad (7)$$

Theorem 1: $D_{QT}(n)$, the average number of total slots for recognizing n tags under QT, is as

$$D_{QT}(n) = 1+\sum_{k=1}^{\infty} 2^{k+1}\left\{1-\left(1-\frac{1}{2^k}\right)^n - n\frac{1}{2^k}\left(1-\frac{1}{2^k}\right)^{n-1}\right\}$$

Proof: Since QT splits the set of colliding tags into two subsets according to the tag IDs, all nodes in the query tree have a degree of either two or zero. All intermediate nodes in the tree also correspond to the collision slots, and each leaf node corresponds to either an idle slot or a readable slot. Therefore,

$$D_{QT}(n) = 1+2C_{QT}(n) = 1+2\sum_{k=1}^{\infty} C_{QT}(n,k). \qquad (8)$$

Substituting Equation 7 into Equation 8 obtains Theorem 1.

4.2. SBA

Let $D_{SBA}(T_{i+1}|T_i)$ be the average number of total slots required by SBA in recognizing T_{i+1} after having recognized T_i, where $|T_i|=n$. $D^*_{SBA}(T_{i+1}|T_i)$ denotes the optimal $D_{SBA}(T_{i+1}|T_i)$. First, suppose that γ arriving tags respond in the first phase. Considering that β tags leave, there are $(n-\beta)$ queries where one staying tag responds and β queries where no staying tag replies in the first phase. For γ arriving tags in which the IDs are uniformly distributed from all combinations of queries, the probability of x arriving tags among γ arriving tags interrogated by a query is $\binom{\gamma}{x}\left(\frac{1}{n}\right)^x\left(1-\frac{1}{n}\right)^{\gamma-x}$. If collisions occur, SBA uses QT to solve them. Thus, the average number of slots in the first phase is as follows:

$$(n-\beta)\sum_{x=0}^{\gamma}\binom{\gamma}{x}\left(\frac{1}{n}\right)^x\left(1-\frac{1}{n}\right)^{\gamma-x} D_{QT}(1+x) + \beta\sum_{x=0}^{\gamma}\binom{\gamma}{x}\left(\frac{1}{n}\right)^x\left(1-\frac{1}{n}\right)^{\gamma-x} D_{QT}(x). \qquad (9)$$

In the second phase, for $\alpha-\gamma$ arriving tags, SBA may estimate $\hat{N}_2$ arriving tags, thus giving $\hat{N}_2$ queries that allow $\alpha-\gamma$ arriving tags to respond. Therefore, the probability of x arriving tags among $\alpha-\gamma$ arriving tags interrogated by a query is $\binom{\alpha-\gamma}{x}\left(\frac{1}{\hat{N}_2}\right)^x\left(1-\frac{1}{\hat{N}_2}\right)^{\alpha-\gamma-x}$. Thus, the average number of slots required by SBA when γ and $\alpha-\gamma$ arriving tags respond in the first phase and in the second phase, respectively, which is denoted $D^{\gamma}_{SBA}(T_{i+1}|T_i)$, is as follows:

$$D^{\gamma}_{SBA}(T_{i+1}|T_i) = (n-\beta)\sum_{x=0}^{\gamma}\binom{\gamma}{x}\left(\frac{1}{n}\right)^x\left(1-\frac{1}{n}\right)^{\gamma-x} D_{QT}(1+x) + \beta\sum_{x=0}^{\gamma}\binom{\gamma}{x}\left(\frac{1}{n}\right)^x\left(1-\frac{1}{n}\right)^{\gamma-x} D_{QT}(x) + \hat{N}_2\sum_{x=0}^{\alpha-\gamma}\binom{\alpha-\gamma}{x}\left(\frac{1}{\hat{N}_2}\right)^x\left(1-\frac{1}{\hat{N}_2}\right)^{\alpha-\gamma-x} D_{QT}(x). \qquad (10)$$

Theorem 2: According to the probabilities P_1 and P_r to determine γ arriving tag, the optimal average number of slots under SBA can be expressed as:

$$D^*_{SBA}(T_{i+1}|T_i) = \sum_{\gamma=0}^{\alpha}\binom{\alpha}{\gamma}(P_1P_r)^{\gamma}(1-P_1P_r)^{\alpha-\gamma} \left\{(n-\beta)\sum_{x=0}^{\gamma}\binom{\gamma}{x}\left(\frac{1}{n}\right)^x\left(1-\frac{1}{n}\right)^{\gamma-x} D_{QT}(1+x) + \beta\sum_{x=0}^{\gamma}\binom{\gamma}{x}\left(\frac{1}{n}\right)^x\left(1-\frac{1}{n}\right)^{\gamma-x} D_{QT}(x) + (\alpha-\gamma)\sum_{x=0}^{\alpha-\gamma}\binom{\alpha-\gamma}{x}\left(\frac{1}{\alpha-\gamma}\right)^x\left(1-\frac{1}{\alpha-\gamma}\right)^{\alpha-\gamma-x} D_{QT}(x)\right\}.$$

Proof:

The probability that arriving tags are interrogated in the first phase is P_r, and the probability that arriving tags determine to respond in the first phase is P_1. Therefore, the probability of γ arriving tags among α arriving tags responding in the first phase is $\binom{\alpha}{\gamma}(P_1P_r)^{\gamma}(1-P_1P_r)^{\alpha-\gamma}$. Thus, the average number of total slots required by SBA is as follows:

$$D_{SBA}(T_{i+1}|T_i) = \sum_{\gamma=0}^{\alpha}\binom{\alpha}{\gamma}(P_1P_r)^{\gamma}(1-P_1P_r)^{\alpha-\gamma} D^{\gamma}_{SBA}(T_{i+1}|T_i). \qquad (11)$$

Equation 11 gives the optimal value when the estimation is perfect, i.e., $\hat{N}_2 = \alpha-\gamma$. Hence, by substituting $\hat{N}_2$ with $\alpha-\gamma$ and substituting Equation 10 into Equation 11, we can obtain the optimal $D^*_{SBA}(T_{i+1}|T_i)$ as Theorem 2.

In Theorem 2, P_r is unknown. However, according to Equation 1, P_r is calculated from the queries in Q. However, we only discern that the number of readable queries stored in Q is n, i.e., n tags recognized in the last frame. Therefore, the number of idle queries should be

derived, and thus, we first compute the total number of required slots when recognizing n tags.

First, the probability of a frame using l slots to recognize n tags is defined as:

$$P_{l,n} \triangleq \Pr\{\text{total slots} = l | n \text{ tags}\}.$$

To obtain $P_{l,n}$, we use the probability generating function as:

$$Q_n(z) \triangleq \sum_{i=0}^{\infty} P_{i,n} z^i. \tag{12}$$

When a collision happens, two 1-bit longer queries are generated after this collision query. Thus, we can obtain the following recursion on the frame size

$$Q_n(z) = \sum_{h=0}^{n} B_h^n Q_h(z) Q_{n-h}(z) z, \quad n = 2, 3, \ldots\ldots \tag{13}$$

where

$$B_h^n \triangleq \Pr\{h \text{ tags in the first subset} | n \text{ tags}\} = \binom{n}{h} 2^{-n}, \tag{14}$$

from the binomial probability of a $(h, n - h)$ split. By differentiating z of Equation 13 by l times and setting $z = 0$, we obtain the probability, $P_{l,n}$, as follows:

$$P_{l,n} = \frac{Q_n^{(l)}(z)|_{z=0}}{l!}. \tag{15}$$

$P_{l,n}$ is not only the probability, wherein the frame has l total slots, but it also represents the probability wherein the frame has n readable slots, $(l - 1)/2$ collision slots, and $l - n - (l - 1)/2$ idle slots. Thus, with this probability, we can get the average number of idle queries, $\bar{n}_I$, as:

$$\bar{n}_I = \sum_{l=0}^{\infty} P_{l,n} \times n_I, \text{where } n_I = (l+1)/2 - n. \tag{16}$$

Since the readable queries and idle queries are randomly scattered in the leaf nodes of the query tree, thus the average P_r can be approximated as:

$$P_r = \frac{n}{n + \bar{n}_I}. \tag{17}$$

Finally, substituting Equation 17 into Theorem 2, the average number of total slots required by SBA can be obtained.

5. Performance comparison

The special case of SBA with $P_1 = 0$ is a blocking algorithm that uses the number of arriving tags existing in the last frame to estimate the number of arriving tags existing in the current frame. Thus, the performances of SBA with $P_1 = 0$ and $P_1 > 0$ should be individually compared with that of AQS. Here, 'SBA0' represents SBA with $P_1 = 0$ while 'SBA+' represents SBA with $P_1 > 0$. The QT is excluded from the comparison because it performs poorly even in comparison with AQS [19-21], and the lines of the other methods in the figures are difficult to view when including QT. The evaluation also excludes ECS because it cannot provide perfect accuracy. Three metrics, the number of collision slots, the number of idle slots, and the number of total slots, are considered when evaluating the efficiency of tag identification. Total slots include collision slots, idle slots, and readable slots. The number of total slots signifies the delay in identifying all tags. The two other metrics, the number of bits sent by the reader and the number of bits sent by all tags, are also considered. More bits means more overhead, causing more power consumption.

First, a simulation is performed to investigate how the numbers of staying tags and arriving tags affect the performance of AQS, SBA0, and SBA+. Here, the number of arriving tags is assumed to be estimated accurately by SBA, i.e., it obtains the optimal performance. This assumption simplifies the observation of trends and the performance comparison.

A more realistic simulation under a mobile environment where tags move within an area is then performed. Unless otherwise specified, SBA0 in the simulation uses an exponential average with a default factor $z = 0.5$, and SBA+ adopts a default probability $P_1 = 0.2$. The effects of some parameters, including the number of tags, the tag moving velocity, and the stationary tag probability, are also investigated. Finally, we investigate the effect of an incorrectly estimated number of arriving tags in SBA under the stationary tag probability following the beta distribution. The influence of P_1 on SBA performance is also studied in this environment.

5.1. Impact of staying tags and arriving tags

Let N be the number of all the tags in the simulation area, T_i be a set of tags existing in f_i, and $|T_i|$ be the number of these tags. The performance of the i+1-th frame, f_{i+1}, can then be investigated by varying the *staying ratio* r_s and the *arriving ratio* r_a, where r_s is the ratio of the number of staying tags over $|T_i|$ and r_a is the ratio of the number of arriving tags over $N - |T_i|$. Two scenarios are considered according to the values of r_s and r_a. In both scenarios, N is 1000 and $|T_i|$ is 500. In scenario 1, r_s is fixed at 0.5, and r_a is varied from 0 to 1. In scenario 2, r_a is fixed at 0.5, and r_s is varied from 0 to 1.

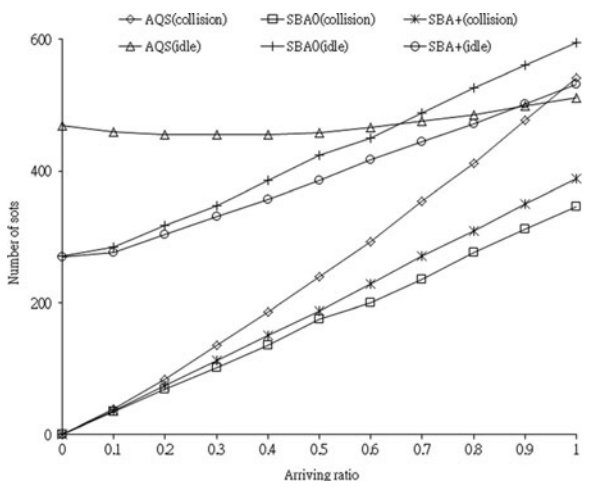


Figure 2 Number of slots versus the arriving ratio r_a when the staying ratio r_s is fixed at 0.5.

Figure 2 depicts the simulation results in scenario 1 and shows that AQS, SBA0, and SBA+ increase identification delay in proportion to the increase in arriving tags since more collision slots, idle slots, and readable slots are caused by them. The SBA0 has fewer collision slots compared to AQS. This is because the former not only avoid collisions between staying tags but also prevent arriving tags from colliding with staying tags. This figure also shows that SBA+ has more collision slots than SBA0 does since it permits a minority of arriving tags to collide with staying tags. Moreover, SBA+ generates fewer collision slots than AQS does because it prevents most of the arriving tags from colliding with staying tags.

Figure 2 shows that AQS has more idle slots compared to SBA0 when $r_a < 0.65$ since AQS transmits many idle queries obtained from the last frame to identify arriving tags, but only a few arriving tags match these queries. In AQS, a larger r_a increases the number of arriving tags that can respond with idle queries caused from the

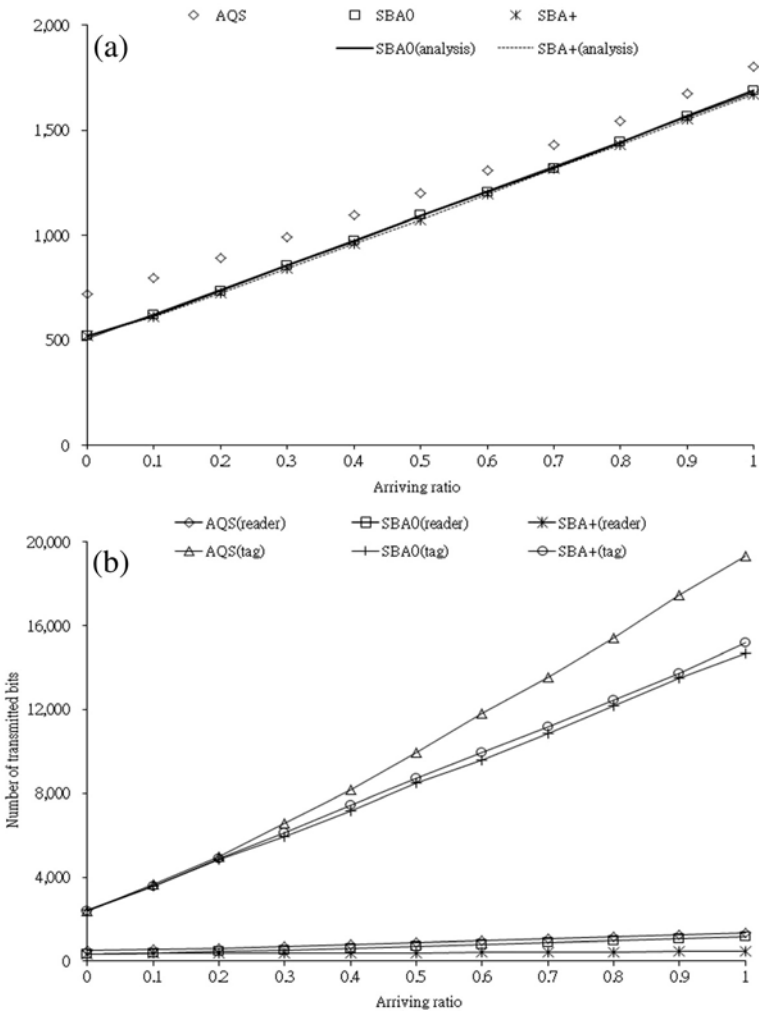


Figure 3 Performance versus the arriving ratio r_a when the staying ratio r_s = 0.5. (a) Number of total slots. **(b)** Number of transmitted bits.

leaving tags or idle queries recorded in the last frame. Therefore, AQS has fewer idle slots compared to SBA0. The SBA+ also generates fewer idle slots compared to SBA0 because SBA+ permits some arriving tags to respond with the idle queries caused from the leaving tags. Finally, Figure 3a shows that SBA0 and SBA+ have similar identification delays and that both outperform AQS in terms of the number of total slots. Moreover, analytical and simulation results quite match.

From Figure 3b, SBA0 and SBA+ obviously outperform AQS in the numbers of bits sent by the reader and sent by all tags. The bits sent by the reader between the former and the latter in a collision slot, an idle slot, and a readable slot, are similar. Since both SBA+ and SBA0 have fewer slots than AQS, they must have less bits sent by the reader than AQS. For the same reason, SBA+ and SBA0 have less bits sent by all tags. On the other hand, SBA+ has slightly less bits sent by the reader and slightly more bits sent by all tags than SBA0. The main reason is that SBA+ generates more collision slots and less idle slots than SBA0 where no bit is transmitted in an idle slot and the bits of ID length are transmitted in a collision slot.

Figure 4 illustrates the simulation results for scenario 2. When $r_s > 0.23$, SBA0 and SBA+ have substantially fewer collision slots compared to AQS. In AQS, when more staying tags are present, arriving tags will more likely collide with them, resulting in the worst performance. However, when $r_s < 0.23$, in AQS, many idle queries caused from the leaving tags are then used by the arriving tags, resulting in fewer collision slots compared to SBA0 and SBA+.

Figure 4 shows that the number of idle slots is expected to decrease as r_s increases. The SBA+ clearly has fewer idle slots than AQS does. The reason is in order to identify arriving tags, AQS must transmit the idle queries recorded in the last frame. If no arriving tags match them in the current frame, these idle queries still cause the idle slots again. Interestingly, the gap between the number of idle slots in AQS and in SBA+ increases as r_s increases. The main reason is that r_s correlates positively with the number of idle queries recorded in the last frame. Thus, for these queries, more idle slots appear in the current frame under a fixed r_a. When $r_s > 0.32$, SBA0 has fewer idle slots compared to AQS for the same reason observed in SBA+. When $r_s < 0.32$, i.e., when the number of leaving tags is large, SBA0, which is a blocking algorithm, prevents arriving tags from responding with idle queries caused from leaving tags. Thus, SBA0 produces more idle slots compared to AQS. Finally, as shown in Figure 5a, in most cases, both SBA0 and SBA+ outperform AQS in terms of the number of total slots. Also, analytical and simulation results quite match. However, when r_s is low, SBA0 requires a longer identification compared to AQS because it generates more collision slots and idle slots.

From Figure 5b, SBA0 and SBA+ obviously outperform AQS in the numbers of bits sent by the reader and sent by all tags. The reason is the same as the descriptions for Figure 3b. Also, the results of comparing SBA+ with SBA0 in Figure 5b are similar to those in Figure 3b for the same reason.

5.2. Performance under a mobile environment

A mobile environment is established as follows:

- *Simulation area*: 10 m × 10 m
- *Identification range of the reader*: 3 m

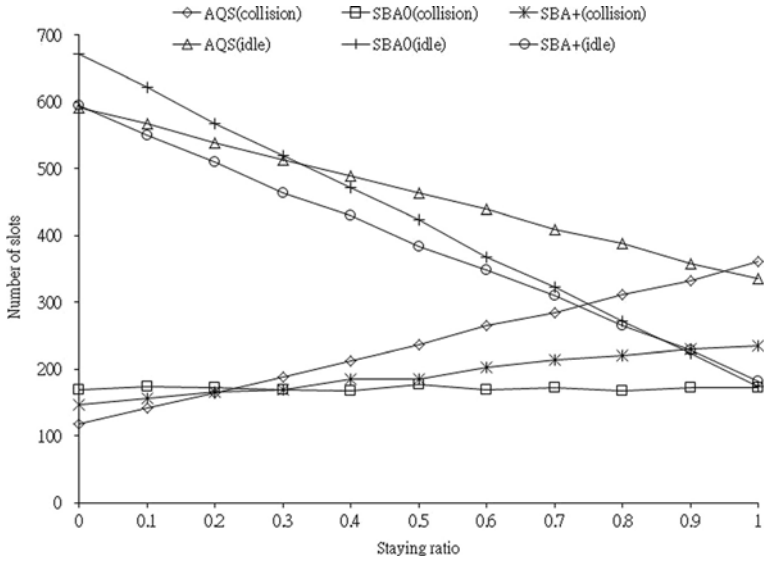


Figure 4 Number of slots versus the staying ratio r_s when the arriving ratio r_a is fixed at 0.5.

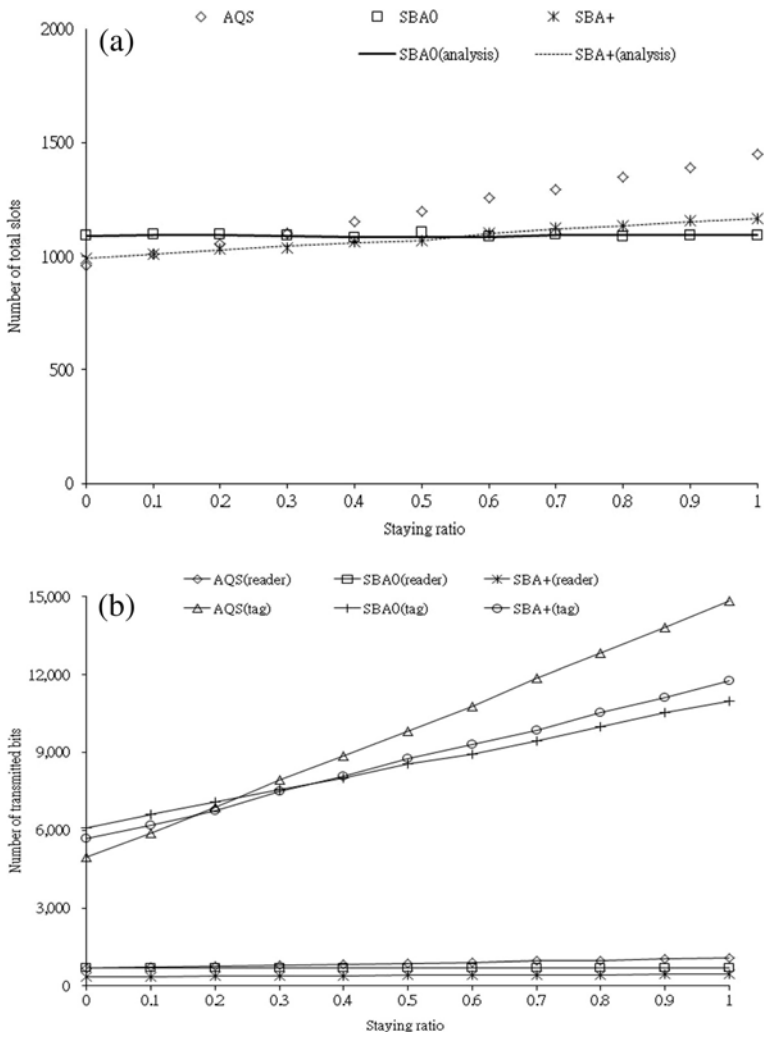


Figure 5 Performance versus the arriving ratio r_s when the staying ratio r_a = 0.5. (a) Number of total slots. **(b)** Number of transmitted bits.

- *Tag ID*: Randomly selected 96-bit ID
- *Number of tags* (N): 1,000
- *Tag moving velocity* (v): 2 m/frame
- *Tag stationary probability* (p_s): 0.5
- *Weight factor* (z): 0.5
- *Probability factor* (P_1): 0.2

The N tags in this environment are mobile within a 10 m × 10 m area. Each tag has a *stationary probability* p_s to decide whether it will move or not during the period of a frame. The *tag moving velocity* v is the distance that each tag moves during one frame if it moves. Since the identification range of the reader is 3 m, some tags enter and leave the range of the reader as arriving tags and leaving tags, respectively. All initial tag positions and directions are randomly selected in the simulation area. Tags that touch the border of the simulation area randomly change direction. Each simulation is performed for 10^6 frames.

The next subsection discusses the results of simulations performed to investigate how parameters, N, p_s, and v affect the performance of AQS, SBA0, and SBA+.

5.2.1. Impact of the number of tags

As the number of tags increases, the numbers of arriving tags, leaving tags, and staying tags increase. Therefore, the numbers of collision slots, idle slots, and readable slots increase linearly as the number of tags increases. The blocking technique used by SBA0 prevent arriving tags from colliding with staying tags to minimize the collisions. On the other hand, SBA+ semi-blocks most of the arriving tags, so it generates slightly more collision slots than SBA0 does.

Since neither SBA0 nor SBA+ retain idle queries in the last frame, they have fewer idle slots compared to AQS. Moreover, when the number of tags becomes large, AQS clearly shows a performance gap between SBA0 and SBA+ because it sends excessive idle queries

obtained from the last frame. Figure 6 shows that, for the above reasons, SBA0 and SBA+ significantly outperform AQS in terms of identification delay regardless of the number of tags even though they may incorrectly estimate the number of arriving tags in this environment.

5.2.2. Impact of tag mobility

As tags move faster, the possibility that they move in and out of the range of the reader becomes more frequent and implies a larger r_a and smaller r_s. More arriving tags cause more collision slots, while fewer staying tags cause more idle slots. Thus, Figure 7 shows that all the algorithms require more slots at high speed than at low speed, since fewer staying tags exist and more arriving tags appear in the former case. The AQS has more collision slots compared to SBA0 and SBA+ because it uses the non-blocking technique. The AQS also produces more idle slots because idle queries in the last frame are reserved for identifying arriving tags. Figure 7 shows that, regardless of the value of v, SBA0 and SBA+ significantly outperform AQS in the identification delay, even when they may incorrectly estimate the number of arriving tags.

5.2.3. Impact of tag stationary probability

At a high stationary probability, most tags are immobile. Therefore, most tags beyond the range of the reader do not enter, and most tags within the range of the reader do not leave. That is, as the stationary probability increases, r_a decreases and r_s increases. In this case, AQS, SBA0, and SBA+ therefore benefit from remembering the tags in the last frame, resulting in fewer collision slots and idle slots. Since few arriving tags come into the range of the reader when the stationary probability is high, SBA0 and SBA+ have an only small improvement on the number of collision slots over AQS. In contrast, when the stationary probability is high, AQS clearly has more idle slots compared to SBA0 and SBA+ since the idle queries recorded in the last frame are very likely to cause idle slots again in the ongoing frame. Therefore, Figure 8 shows that SBA0 and SBA+ again outperform AQS.

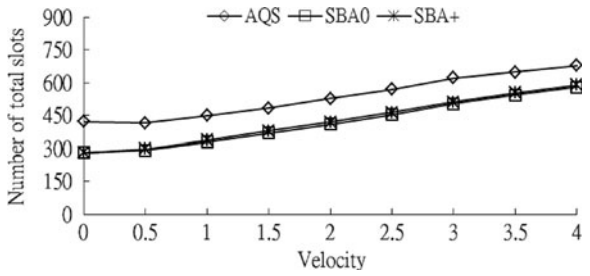


Figure 7 Number of total slots versus the tag moving velocity in a mobile environment.

5.2.4. Performance under an incorrectly estimated number of arriving tags

In the previous subsection, the number of arriving tags may be incorrectly estimated in the steady state. However, these errors are small. This study therefore further examines the effect of p_s following the beta distribution to observe the performance of SBA in the case of an incorrectly estimated number of arriving tags.

In this simulation, the probability of the tags randomly selecting the stationary probability p_s in each frame is determined by the probability density function of the beta distribution [27] with parameters α and β:

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}du}, \quad \alpha,\beta > 0; 0 \le x \le 1.$$

With the beta distribution, different degrees of the variation in the numbers of staying tags and arriving tags can be easily observed in the same figure. Here, $\alpha = \beta$. Therefore, this distribution is U-shaped under $\alpha < 1$ and $\beta < 1$, while it is similar to the normal distribution when $\alpha > 1$ and $\beta > 1$. When $\alpha = 1$ and $\beta = 1$, the distribution is uniform. Thus, smaller α and β imply a larger variation of p_s, which increases the variation in the numbers of arriving tags and staying tags. Thus, this simulation can clearly exhibit the effects of the variation in the

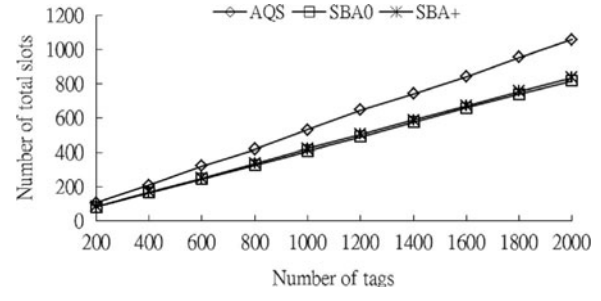


Figure 6 Number of total slots versus the number of tags in a mobile environment.

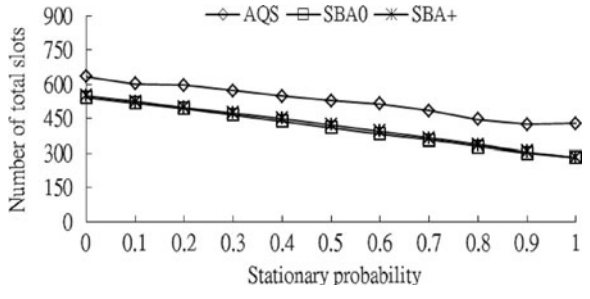


Figure 8 Number of total slots versus the tag stationary probability in a mobile environment.

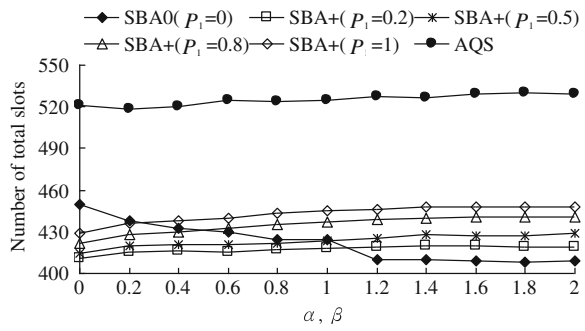


Figure 9 Number of total slots with different P_1 versus p_s that follows the beta distribution.

number of staying tags and arriving tags on the performance of SBA and AQS.

Figure 9 shows the number of total slots of SBA when using five values of P_1, 0 (i.e., SBA0), 0.2, 0.5, 0.8, and 1, and AQS. Regardless of the values of α and β, SBA+ requires a quite stable number of slots because the estimate based on the information obtained in the first phase is sufficiently accurate. Also, when P_1 is smaller, SBA+ performs better because SBA+ allows fewer arriving tags to collide with staying tags and have many chances to use the slots of leaving tags in the first phase. When the number of arriving tags substantially varies, i.e., $\alpha \leq 1$ and $\beta \leq 1$, SBA0, which uses the information of the last frame, generates a larger estimation error and degrades performance. On the other hand, when $\alpha > 1$ and $\beta > 1$, because the variation in the number of arriving tags and staying tags are smooth, SBA0 still has a correct estimation even when it uses the information of the last frame. Furthermore, SBA0 prevents arriving tags from colliding with staying tags, causing that it has better performance than SBA+. In summary, the figure shows that SBA+ with $P_1 = 0.2$ is superior when $\alpha \leq 1$ and $\beta \leq 1$ while SBA0 is superior when $\alpha > 1$ and $\beta > 1$.

Still, SBA+ with $P_1 = 1$ significantly outperforms AQS. The SBA+ is superior because it only transmits readable queries, but not idle queries, obtained from the last frame. Moreover, SBA+ uses proper queries with accurate estimation to interrogate arriving tags. Thus, most idle slots and collision slots in AQS are avoided in SBA+ with $P_1 = 1$.

6. Conclusions

Collisions that occur during simultaneous tag transmissions are a major cause of delayed tag identification in RFID systems. The novel SBA proposed in this study not only exploits information obtained from the last frame for reducing collisions among staying tags but also reduces collisions by blocking most arriving tags, which may collide with staying tags. The SBA also reduces unnecessary idle queries by only sending the readable queries obtained from the last frame. Additionally, SBA quickly identifies arriving tags by estimating their number and generating proper queries based on the information of the first phase in the current frame.

The formal analysis obtains a formula for calculating the number of total slots required in SBA. Several simulations are also performed. The main observations in this study are summarized as follows:

(1) When the numbers of tags in and out of the reader's range are fixed, SBA outperforms AQS in almost all cases except in the case of a small r_s and a large r_a; in which case, AQS allows many arriving tags to match the idle queries caused from the leaving tags.
(2) In a mobile environment, SBA always outperforms AQS regardless of the number of tags, the tag velocity, and the stationary probability. The SBA is superior in terms of collision, idle, total slots, and the transmitted bits.
(3) When the number of arriving tags is generally stable, setting $P_1 = 0$ in SBA obtains the best performance since this number can be accurately estimated based on the information in the last frame. However, setting $P_1 = 0.2$ in SBA obtains the best performance when this number widely varies. Thus, an appropriate P_1 should be set according to the variation in the number of arriving tags.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The authors would like to thank the National Science Council, Taiwan, for financially supporting this research under contract no. NSC 102-2219-E-011-002.

Author details

[1]Department of Information Management, National Taiwan University of Science and Technology, No.43, Sec. 4, Keelung Rd., Taipei 106, Taiwan. [2]Center of Teaching and Learning Development, Nanhua University, No. 55, Sec. 1, Nanhua Rd., Dalin Township, Chiayi County 62249, Taiwan. [3]Department of Computer Science and Information Engineering, Nanhua University, No. 55, Sec. 1, Nanhua Rd., Dalin Township, Chiayi County 62249, Taiwan.

Received: 25 March 2013 Accepted: 9 September 2013
Published: 18 September 2013

References

1. Z Lei, TSP Yum, The optimal reading strategy for EPC Gen-2 RFID anti-collision systems. IEEE Trans. Commun. **58**(9), 2725–2733 (2010)
2. Y Maguire, R Pappu, An optimal Q-algorithm for the ISO 18000-6C RFID protocol. IEEE Trans. Autom. Sci. Eng. **6**, 16–24 (2009)
3. Z Lei, TSP Yum, Optimal framed aloha based anti-collision algorithms for RFID systems. IEEE Trans. Commun. **58**(12), 3583–3592 (2010)
4. YH Chen, SJ Horng, RS Run, JL Lai, RJ Chen, WC Chen, Y Pan, T Takao, A novel anti-collision algorithm in RFID systems for identifying passive tags. IEEE T. Ind. Inform. **6**(1), 105–121 (2010)

5. L Xie, B Sheng, CC Tan, H Han, Q Li, D Chen, Efficient tag identification in mobile RFID systems, in *Proceedings of the IEEE INFOCOM* (San Diego, 2010), pp. 1–9
6. CF Lin, FYS Lin, Efficient estimation and collision-group-based anti-collision algorithms for dynamic frame-slotted aloha in RFID networks. IEEE Trans. Autom. Sci. Eng. **7**(4), 840–848 (2010)
7. Z Xuan, Y Gang, Research and simulate of the optimization anti-collision technology in UHF RFID system, in *the International Conference on Electric Information and Control Engineering* (Wuhan, 2011), pp. 643–646
8. J Park, MY Chung, TJ Lee, Identification of RFID tags in framed-slotted ALOHA with robust estimation and binary selection. IEEE Commun. Lett. **11**(5), 452–454 (2007)
9. MK Yeh, JR Jiang, ST Huang, Parallel response query tree splitting for RFID tag anti-collision, in *the 40th International Conference on Parallel Processing Workshops* (Taipei City, 2011), pp. 6–15
10. Y Jiang, R Zhang, An adaptive combination query tree protocol for tag identification in RFID systems. IEEE Commun. Lett. **16**(8), 1192–1195 (2012)
11. C-K Liang, H-M Lin, Using dynamic slots collision tracking tree technique towards an efficient tag anti-collision algorithm in RFID systems, in *the Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing* (Fukuoka, 2012), pp. 272–277
12. H Gou, J Yang, Y Yoo, A hybrid technique based query tree protocol for energy efficiency in RFID system, in *the 12th International Conference on Computer and Information Technology* (, Chengdu, 2012), pp. 82–87
13. JH Choi, D Lee, H Lee, Query tree-based reservation for efficient RFID tag anti-collision. IEEE Commun. Lett. **11**(1), 85–87 (2007)
14. L Pan, H Wu, Smart trend-traversal: a low delay and energy tag arbitration protocol for large RFID systems, in *IEEE INFOCOM* (Rio de Janeiro, 2009), pp. 2571–2579
15. MK Yeh, JR Jiang, ST Huang, Adaptive splitting and pre-signaling for RFID tag anti-collision. Comput. Commun **32**(17), 1862–1870 (2009)
16. W Zein-Elabdeen, E Shaaban, An enhanced binary tree anti-collision technique for dynamically added tags in RFID systems, in *the 2nd International Conference on Computer Engineering and Technology* (Chengdu, 2010), pp. 349–353
17. X Chen, Y Yao, G Liu, Y Su, Y Chen, S Miao, IBTA: an IBT-tree based algorithm for RFID anti-collision, in *the International Forum on Information Technology and Applications* (, Kunming, 2010), pp. 407–410
18. DK Klair, KW Chin, R Raad, A survey and tutorial of RFID anti-collision protocols. IEEE Commun. Surv. Tut. **12**, 400–421 (2010)
19. J Myung, W Lee, *Adaptive binary splitting: a RFID tag collision arbitration protocol for tag identification*, vol. 5, 11th edn. (Mobile Networks and Applications, Springer, Netherlands, 2006), pp. 711–722
20. J Myung, W Lee, TK Shih, An adaptive memoryless protocol for RFID tag collision arbitration. IEEE T. Multimedia **8**(5), 1096–1101 (2006)
21. J Myung, W Lee, J Srivastava, TK Shih, Tag-splitting: adaptive collision arbitration protocols for RFID tag identification. IEEE T. Parall. Distr. Syst. **18**(6), 763–775 (2007)
22. YC Lai, CC Lin, A pair-resolution blocking algorithm on adaptive binary splitting for RFID tag identification. IEEE Commun. Lett. **12**(6), 432–434 (2008)
23. YC Lai, CC Lin, Two blocking algorithms on adaptive binary splitting: single and pair resolutions for RFID tag identification. IEEE/ACM Trans. Network. **17**(3), 962–975 (2009)
24. YC Lai, CC Lin, Two couple-resolution blocking protocols on adaptive query splitting for RFID tag identification. IEEE Trans. Mob. Comput. **11**(10), 1536–1233 (2012)
25. B Sheng, L Qun, W Mao, Efficient continuous scanning in RFID Systems, in *IEEE INFOCOM* (San Diego, 2010), pp. 1–9
26. Technical Committee ISO/IEC JTC 1, *Information technology - radio frequency identification for item management. Part 6: parameters for air interface communications at 860 MHz to 960 MHz: amendment 1: extension with type C and update of types A and B, ISO/IEC 18000–6:2004/Amd. 1:(E)* (ISO/IEC, Geneva, 2006)
27. M William, WD Dennis, SL Richard, *Mathematical Statistics with Applications*, 4th edn. (PWS-KENT Pub. Co., Riverside, 1990), pp. 171–172

doi:10.1186/1687-1499-2013-231
Cite this article as: Lai *et al.*: A semi-blocking algorithm on adaptive query splitting for RFID tag identification. *EURASIP Journal on Wireless Communications and Networking* 2013 **2013**:231.