Smart Batch-Processing Protocol for RFID Multiple Tag Identification

XIAOJIAO YANG, XINXIN LIU, AND BIZAO WU

1School of Information Science and Engineering, Chongqing Jiaotong University, Chongqing 400074, China
2Solution Center, China Mobile IoT Company Limited, Chongqing 401121, China

Corresponding author: Xinxin Liu (lxx@cqjtu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61903053, and in part by the Chongqing Municipal Education Commission Research Program under Grant KJQN201900702.

ABSTRACT The common paradigm of the RFID network is to obtain data from multiple tags through a reader. The tag collision problem seriously affects the efficiency of tag identification. The anti-collision algorithm provides a reliable guarantee to identify multiple tags as fast as possible. Many papers have proposed query tree algorithms based on bit tracking, but these algorithms still have problems such as idle paging and redundant prefixes. In this paper, we start with an analysis of state-of-the-art anti-collision algorithms. Based on summarizing current works, we propose a Smart Batch-Processing protocol (SBP) for RFID multiple tag identification based on \( M \)-ary collision bit arbitration mechanism. In the SBP, the reader firstly maps \( m \)-bit collision bits into \( 2^m \) slots and then predicts the distribution of tags in slots. Finally, the reader allows the tag to return the remaining part of its ID at the mapped slot using a batch-processing command. By using SBP, the reader can simultaneously identify \( 2^m \) groups of tags in a slot. Besides, empty queries can be eliminated in the tag identification process. The description of SBP is given first, and then, its performance is theoretical analyzed and simulated. Both theoretical analysis and extensive simulation results show that the SBP algorithm reduces 22.94% and 22.28% of time and energy consumption compared to other \( M \)-ary collision bit arbitration based methods.

INDEX TERMS Anti-collision, batch-processing, \( m \)-ary, radio frequency identification, time efficient.

I. INTRODUCTION Recently, with the rapid development of Internet of things (IoT), Integrated circuits (IC), and communication networks, and the gradual reduction in application costs of RFID systems, RFID has been widely used in various applications including inventory control and supply chain management. A typical RFID system usually includes an interrogator (also named reader) and several transponders (also named tags). When the reader probes the tags within its coverage, it may happen that multiple tags simultaneously respond to the reader, which is called a tag collision problem. The tag collision will cause the reader can not recognize the information returned by the tags without an anti-collision algorithm [1], and thus fail to successfully identify any tags.

To cope with the above challenges, a variety of anti-collision or tag identification algorithms [1]–[7] have been proposed. The existing anti-collision algorithms are mainly divided into three categories: Aloha-based, hybrid-based, and tree-based. Many algorithms based on dynamic framed slotted Aloha (DFSA) have been presented in [1]–[7]. The main idea of them is that the reader firstly broadcasts the frame length to the tags (i.e., the number of available slots for a tag to choose), and a tag randomly selects a slot reply to its ID. According to the slot statistics, the reader dynamically adjusts the frame length. The DFSA algorithm can achieve a maximum system efficiency of 36.8% when the number of tags is equal to the number of available slots. In practice, the reader does not know a priori information of the number of tags, so it can only estimate the number of tags based on the number of collisions, success, and empty slots counted in the frame. The more accurate the estimated result, the higher the efficiency of the algorithm, but the higher the computational complexity [2]. The advantage of the Aloha-based algorithm is that it does not require bit-level synchronization technology, so it is suitable for the UHF RFID system. The disadvantage is that the tag has a large implementation cost and the algorithm suffers from tag starvation problem. The
hybrid algorithms have been proposed in those literatures [6]–[12], which combine the characteristics of Aloha-based and tree-based algorithms. The core idea of such type of algorithm is that when the reader detects a collision, it precedes the collided tag set using a tree-based algorithm (the CNCAA algorithm proposed in [11] which uses both collided and non-collided parts of information in collision slots for tag’s identification.). Although the system efficiency of the hybrid algorithm is higher than that of Aloha-based algorithms. For instance, the system efficiency reported in [9] can reach 48.5%: in the paper [12], the efficiency of this algorithm reached 73% by using the information available in the collision time slot, called conflict data, to uniquely identify labels. However, it does not guarantee that all the tags are identified. The implementation complexity of hybrid algorithms is also higher than Aloha-based algorithms. Besides, because of the need to use bit identification and tracking technology, it is not suitable for UHF RFID systems [6].

Tree-based algorithms [8], [13]–[23] require that when multiple tags send IDs simultaneously, the reader can accurately locate individual collided bits of tag ID string. Specifically, in ISO-14443/15693, the reader uses the Manch-ester code to identify the bit value of the ID string. The main idea of a tree-based algorithm is to maintain the string before the highest collision bit as the common prefix when the reader detects a collision, and then divides the collided tag set according to the collision bits. Some algorithms [8], [13]–[15] divide the collided tag set into two groups according to the highest collision bit. Many algorithms [16]–[21] divide the collided tags into M groups. There are still some algorithms [22], [23] adopting new grouping methods. In essence, a tree-based algorithm continuously separates collided tags into smaller groups until there is only one tag in the group. The main evaluation metrics for measuring the performance of the tree-based algorithms are time and energy consumption, which depend on the number of interactions between the reader and tags and the number of transmitted bits [24]. Many newly proposed algorithms focus on these two aspects.

In [13], CwT is proposed to reduce the number of transmitted bits between the reader and tags. The parameters sent by the reader to the tags include query prefix and w. After receiving the query command from the reader, the tag only returns a partial ID of length w instead of the remaining ID. However, if the sum of the length of w and the prefix is not equal to the ID length, the reader needs to send the query command again to obtain the complete ID. Thus, the number of interactions has been increased. In STT [23], when detecting a collision, the reader increases the length of the prefix, thereby skipping the redundant collision queries. When detecting an empty slot, the reader shortens the length of the prefix, thereby skipping the redundant empty queries. The advantage of the STT algorithm is to reduce the number of interactions. The demerit is that there are many empty slots. The MQT algorithm proposed in [19] defines a set of mapping rules (original m bits is mapped into $2^m$ bits). When the reader receives the mapped string returned by multiple tags simultaneously, it can recover the original data according to the mapping tables. The MQT can reduce the number of interactions between the reader and the tags. However, the number of transmitted bits has been increased due to the combined transmission of mapped data and the remaining ID. The NAA algorithm proposed in [16], which only uses the collision bits to generate a new tag’s temporary ID and the followed operation only for this new ID. The NAA algorithm’s processing of the new tag ID is mapping first transmitting later. When a collision occurs, NAA can recover the original data according to the mapping rules (original m bits is mapped into $2^m$ bits). The NAA can reduce the times of interactions between the reader and the tags. However, the amount of transmitted bits has been increased due to the combined transmission of mapped data and the remaining ID. The literature [17] proposes an MCT algorithm. When a collision occurs, the reader broadcasts the position of m-bit collision bit and the tags determine which slot to respond according to the value of collision bits. The merit of MCT is that the reader can potentially identify up to $2^m$ tags by preprocessing the $m$-bit collision bits. The demerit is that the reader needs to transmit query command at the beginning of each slot, and there are many empty slots.

The above work focuses on decrease the times of interactions between the reader and tags and the number of transmitted bits. However, their performance needs to be further improved because of a redundant prefix, unused collided bits, and idle paging.

This paper proposes a new algorithm namely SBP based on the $M$-ary collision bit arbitration mechanism. In SBP, when a collision occurs, the reader first detects the actual distribution of $m$-bit collision bits through a BQ command (After receiving the BQ command, the tag will generate $2^m$ mapped string based on the value of original m bits). The reader then sends a batch-processing command CMD to allow the tags to know the detected results from the BQ command. The merits of SBP are two-fold. For one thing, the reader can predict the actual distribution of collision bits through BQ command, thereby effectively eliminating a large number of empty queries in traditional $M$-ary algorithms. For another, the tag responds to the reader in the corresponding slot based on the values of its collision bits, so that the reader can batch process the tags. Theoretically, the reader can recognize up to $2^m$ tags in a slot by sending a CMD command, which greatly reduces the number of queries.

The remainder of this paper is organized as follows. Section II introduces some preliminaries for our work. The basic idea and description of SBP are described in Section III. Section IV makes a theoretical analysis of the number of queries, the total time consumption, and the energy consumption of SBP. The simulation results are provided in Section V. Section VI summarizes this paper.
II. PRELIMINARY

We first introduce the basic principle of the $M$-ary collision bit arbitration mechanism. The reader will list $2^m$ possible outcomes of the highest $m$-bit collision bits and then divide the collided tags into $2^m$ groups. For example, when $m = 2$, four prefixes will be generated, $pre = 00$, 01, 10, or 11. And the collided tags will be divided into four groups. In the $M$-ary-based algorithms, the larger the $m$ value, the more the number of groups. In order to achieve a compromise between the number of collision slots and empty slots, the value of $m$ is usually set to 2 or 3 [18]–[20].

To reduce the number of interactions between the reader and tags, the SBP will batch-process the tags. In SBP protocol, multiple tags are identified by a reader at a time slot. Therefore, it is necessary to temporarily store tag ID information using a queue. The basic working principle of the queue is introduced as follows.

The queue is a special linear table that defines the operation of insertion and deletion. Specifically, the operation of inserting and deleting data into the queue can only be performed at the head of the queue and at the tail of the queue, respectively. In the proposed SBP algorithm, the reader maintains two queues which are Pque used to store prefixes and Bque used to store parameters of BQ command, respectively. We will introduce the usage of these two queues in the following sections.

The data communication between the reader and the tags is based on Manchester coding, that is, data-0 and data-1 are identified through the transition of high and low levels. The reader can determine whether a collision has occurred based on the level conversion. It is worth noting that the SBP algorithm proposed in this paper is based on the strict synchronization of the data returned by the tags. The link timing diagram of the SBP algorithm is described in Fig. 1.

As observed in Fig. 1, there are two types of commands namely BQ and CMD, respectively. In the proposed SBP algorithm, the reader main-
CMD($B_S, B_L$): It is a batch processing command. $B_S$ indicates the distribution of slots which is calculated from the reply of previous BQ command. $B_L$ is the length of the data string returned by the tag for the current CMD command. The value of $B_S$ is the string obtained by setting the collision bit as 1 received by the last BQ command and unchanging the non-collision bits. For example, when $B_S = 0001001$, it means that the data will be transmitted between reader and tag only in the first and fourth slots and the remaining time slots are the idle slots. It is noted that when $B_S = \epsilon$ and $B_L = L$, the tags will respond to the full ID to the reader.

Pque: The queue is used to store prefixes in a first-in-first-out way. It can only insert the data at the tail of the queue and delete the data at the head of the queue. The process of popping out the queue is to read the first value from the queue and delete it from the queue, and then the previous second value becomes the new header of the queue. After receiving the response to the BQ command, the reader will update the value of Pque.

Bque: The queue is used to store the parameters of the BQ command. The queue’s operation is the same as Pque. After receiving the response to the CMD command, the reader will generate a set of BQ command parameters and insert them into the tail of Bque if the recovered ID collides.

RegM: The $M$-bit register in the tag is used to store the mapped data of $m$-bit collision bits. After receiving the BQ command, the tag will update the value of RegM.

### B. READER OPERATION

This section mainly introduces the proposed SBP algorithm in detail. Tab. 2 describes the mapped tables for $m = 2$ and 3.

#### TABLE 1. Character meaning.

| Symbol | Type         | Meaning                                                                 |
|--------|--------------|-------------------------------------------------------------------------|
| $\parallel$ | operators   | Connector operators, for example 0001$\parallel$1100$\parallel$000111100 |
| $<<$  | operators    | Left shift operators, for example 0001$<<$2$\parallel$0100            |
| $^*$  | operators    | Calculate the length of the $*$ string                                 |
| $L$  | Decimal      | The length of tag’s ID.                                                 |
| $m$  | Decimal      | The collision bits of SBP algorithm can handle at a time               |
| $M$  | Decimal      | The binary number with length $M \times 2^n$ can be obtained after the $m$-bit binary number is mapped |
| $rID$ | String       | The position of the i-th collision bit of the string                    |
| $C(i)$ | Decimal      | Where $C(i)$ represents the position of the highest collision bit of $rID$ |
| $rID[i]$ | String       | The i-th string of $rID$, if $i > j$ then $rID[i,j] = null$             |
| $Pp$  | String       | The time slot detection result received by the reader                   |
| $S(i)$ | Binary       | The position of the i-th collision bit in $rS$, where $S(i) = m$, for example, if $rS = 000010000$, then $S(1) = 001$, $S(2) = 100$ |

Initially, the reader initializes the parameters $B_S = \epsilon$, $B_L = L$, $Pque = null$, and $Bque = null$.

1) IDENTIFICATION STARTING: THE READER BROADCASTS CMD($B_S, B_L$) COMMAND

#### a: THE GENERATING PROCESS OF BQ COMMAND PARAMETERS

After receiving the tag’s response $rID$, the reader will directly identify the tag if no collision is detected. Otherwise, the reader will generate a set of BQ command parameters ($P_s$, $P_C$, $P_{P(2)}$, ..., $P_{P(m-1)}$) and insert them into the Bque. The generating process of BQ command parameters is as follows. $P_s$ means the string before the highest collision bit. If the highest collision bit is the first bit, then $P_s = \epsilon$. Otherwise, $P_s = [1:rC(1)-1]$. $P_c$ denotes the collision segment, which can be obtained by setting the collision bits $C(1), C(2), ..., C(m)$ to 1. $Pp(i)$ denotes the position of the i-th collision bit. As described in Fig. 2(b), when the reader receives $rID = xxxxxxxxx$, the first bit is the highest collision bit and $P_s = \epsilon$. In Fig. 2(b), the first three collision bits are $C(1), C(2)$ and $C(3)$, and $rID[1:3] = xxx$ which are marked with red color, then, $Pc = 111$. Since $m = 3$, the BQ command parameters generated by $rID$ is $(\epsilon, 111, 2)$.

2) DETECTING THE DISTRIBUTION OF TAGS IN COLLISION SLOTS

The reader determines whether the Bque is null. If it is null, it indicates that all the tags within the reader’s vicinity have been identified and the identification process ends. Otherwise, the reader pops out a set of parameters from Bque as the parameters of the BQ command. Then, the reader broadcasts the slot probing command BQ($P_s$, $P_C$, $P_{P(2)}$, ..., $P_{P(m-1)}$) to the tags.

#### a: PREFIX COMPOSING AT THE READER SIDE

When receiving the tag response $rS$, the reader labels the position of $rS$, then generates a series of prefixes and pushes them into Pque. The calculation method of prefix is as follows: $pre(i) = P_s|[S(i)[1]]|P_C[2: Pp(2)-1]|...|P_C[Pp(j-1)-1: Pp(j)-1]|S(i)[j]|...|P_C[Pp(m-1)+1: |P_C[1-1]|S(i)[m]]$, where $1 \leq j \leq m$. As described in Fig. 2(b), the response to

### TABLE 2. Mapping table of $M(i)$ and $M(i)$ when $m = 2$ or 3.

| $m$ | $M=2^m$ | $M(i)$ |
|-----|---------|--------|
| 2   | 4       |        |
| 3   | 8       |        |
BQ(ε, 111, 2) is $rS = X0X00X0$. The positions of collision bits in $rS$ are (S(0), S(2), S(5), S(7)) = (000, 010, 101, 111).

According to $S(0) = 000$, $P_s = \varepsilon$ and $P_2(2) = 2$, we can have $pre(1) = null || 0 || null || 0 || null || 0 = 000$. Similarly, we can also have $pre(2) = 010$, $pre(3) = 101$, $pre(4) = 111$.

### 3) BATCH PROCESSING ROUND

The reader pushes the $pre(1),...,pre(i)$ into Pque and determines whether the value of $B_L = L-(|Ps|+|Pc|)$ is 0. If $B_L = 0$, it means that the current pre is a complete ID, the reader will identify the tag and skip to step 2. If $B_L \neq 0$, it means that the current pre is not a complete ID, the reader will send CMD($B_s$, $B_L$) to allow the tag to respond to the rest ID. As described in Fig. 2(b) step 2, the reader calculates that $B_L = L-(|Ps|+|Pc|) = 8-(0+3) = 5$, after the command BQ, it means that the reader will send CMD($B_s$, $B_L$) to allow the tag to respond to the rest ID. Set the collision bit (which are marked with pink color) in $rS$ to 1 and we can get $B_s = 10100101$.

**a: COMPOSE A COMPLETE TAG’S ID AT THE READER SIDE:**

After receiving the response $rP$ to CMD($B_s$, $B_L$), the reader first decodes $rP$ and then skips to step 2. The decoding process of $rP$ is as follows: the reader extracts ($pre(1),...,pre(i)$) from Pque and calculates IDs $rid(1),...,rid(i)$ based on ($pre(1),...,pre(i)$), $B_L$ and $rP$. Where $rid(i) = pre(i)$ $rP[B_L \times (i-1)+1: iB_L]$. If no collision in $rid(i)$, the reader can identify the corresponding tag. Otherwise, the reader will generate a series of BQ command parameters and push them into Bque. Referring to the 3rd step in Fig. 2(b), when receiving $rP = x0xxx10001011 x1xxx$, the reader first extracts ($pre(1),pre(2),pre(3),pre(4)) = (000,0101,0111,111)$ and then calculates $rid(1) = pre(1) || rP[B_L \times (i-1)+1: iB_L] = 000 || rP[1:5] = 000 || x0xxx = 000X0xx$. Since there is a collision in $rid(1)$, the reader needs to generate BQ command parameters (000, 1011, 3). $pre(2)$ and $rP[6:10]$ are marked with green color in Fig. 2(b). The reader can directly identify T4 due to $rid(2) = 01010001$.

**C. TAG OPERATION**

In the SBP algorithm, there are only two commands received by the tag from the reader: the slot probing command BQ and the batch processing command CMD. In the following, we will introduce the tag’s response to BQ and CMD in detail.

**a: THE PROCESSING PROCESS AFTER THE TAG RECEIVES THE BQ($P_s$, $P_c$, $P_{(2)},...,P_{(m-1)}$) COMMAND**

The tag compares ID[1]: $|P_s|$ and $P_s$ then determines whether to send the value of RegM to the reader. If ID[1]: $|P_s| = P_s$, the tag updates the value of RegM and sends the updated value to the reader. The calculation process of RegM value is as follows. Based on the parameters $P_c$, $P_{(2)},...,P_{(m-1)}$, the
reader calculates \( m(i) = ID[P_S]+1 \) \( || \cdots || ID[P_S]+P_{\rho(i)} \) \( || \cdots || ID[P_S]+P_{C[i]} \), and then obtains \( M(i) \) using Tab. 2. If \( ID[1]: P_S \neq P_S \), the tag will set the register RegM to null and will not reply to the reader.

**b: THE PROCESSING PROCESS AFTER THE TAG RECEIVES THE CMD(Bs, B_L) COMMAND**

The tag extracts the values of \( Bs \) and \( B_L \). If \( Bs = \epsilon \) and \( B_L = L \), the tag will reply to the complete ID to the reader. If \( Bs \neq \epsilon \), the tag judges if RegM value is null. If RegM = null, the tag will not reply to the reader. Otherwise, the tag replies ID[\( L+1-B_L:L \)] to the reader after \( T_d \) bits transmission time delay. The calculation method of \( T_d \) is \( T_d = B_L \times \text{Count}((\text{RegM-1} & Bs)) \), where \( \text{Count}(x) \) represents the number of 1 in binary number \( x \). After the tag T4 receives the third step batch command CMD (10100101,5) in Fig. 2 (b), since the register regM value of the tag \( T_d \) is 0000100 which are marked with blue color, the tag T4 waits for the \( T_d \) bit time interval and returns the remaining data ID, where \( T_d = 5 \times \text{Count}(00000011 & 10100101) = 5 \times \text{Count}(00000001) = 5 \times 1 = 5 \).

**IV. THEORETICAL ANALYSIS**

There are only two types of query commands in the SBP algorithm, named BQ or CMD commands. For ease of understanding, we regard CMD as a frame start command, and the BQ as a slot start command. Taking \( M = 2^m = 4 \) as an example, the reader can tackle two collided bits at a slot, and these two collided bits have four possible outcomes: 00, 01, 10, and 11. For a given slot, it has three states, collision, success, and empty.

**A. NUMBER OF PAGING TIMES**

In the SBP algorithm, when detecting a collision the reader sends a BQ command to detect whether there are empty slots in 2\(^m\) slots. Then the reader sends the detecting results to the tags, ensuring that there is no empty slot when the tag transmits the remainder ID to the readers. The number of times the BQ command transmitted is equal to the number of collision slots. Therefore, the total number of queries can be calculated as:

\[
Tt(n) = Tbg+Tcmd \approx Sc(n)+(1+Sc(n)) = 2Sc(n)+1
\]  

(3)

where \( Sc(n) \) is the number of collision slots that will occur during the identification of \( n \) tags.

The SBP algorithm divides the tag set into \( M(M = 2^m) \) subsets through the batch command CMD, where \( m \) is the number of collision bits. Considering the random distribution of tag IDs, each collided tag set has \( M \) subsets. Define the divided subsets as \( A_j(j = 0, 1, \ldots, M-1) \). Assume the number of tags to be recognized is \( n \), then the probability that \( i \) tags fall into the same subset can be expressed as:

\[
P(|A_i| = i) = C_n^i \left( \frac{1}{M} \right)^i \left( 1 - \frac{1}{M} \right)^{n-i}
\]

(4)

Here in \([*]\) means the number of elements in the set. For ease of analysis, taking \( m = 2 \) as an example, the reader can detect two collision bits. The corresponding subsets of two collision bits are \( A_0 = 00, A_1 = 01, A_2 = 10, \) and \( A_3 = 11 \). Only if the distribution of tags satisfies the following scenarios, the reader can detect two collision bits.

(a) Both \( A_0 \) and \( A_3 \) have at least one tag.

(b) Both \( A_1 \) and \( A_2 \) have at least one tag.

Let \( B_1 = (|A_0| \geq 1) \cap (|A_3| \geq 1) \), \( B_2 = (|A_1| \geq 1) \cap (|A_2| \geq 1) \), we can know that the probability of \( i \) tags falling into the \( A_j \) subset is \( P(|A_0| \geq 1) \cap (|A_3| \geq 1) \). Let \( Sc(n) \) denote the number of collision slots which can be expressed as:

\[
Sc(n) = \begin{cases} 
0, & n = 0 \text{ or } 1 \\
1, & n = 2 \\
1+\sum_{j=0}^{M-1} \sum_{i=2}^{n-1} P(|A_j| = i) B_1 B_2 Sc(i), & n > 2
\end{cases}
\]

(5)

**Proof:** When \( n = 0 \) or 1, it is obvious that the reader can identify the tag by sending only one command. When \( n \geq 2 \), we can calculate \( Sc(n) \) through a recursive method. When \( n \geq 2 \), the tag’s reply to the first CMD command must be a collision slot. The tag’s reply to each batch command CMD is regarded as a data frame, and each frame contains \( M = 2^m \) slots.

\[
P(|A_i| = i B_1 B_2) = \frac{P(B_1 B_2 | A_i = i)}{P(B_1 B_2)}
\]

(6)

Next, let’s calculate the value of \( P(B_1 B_2) \) part in formula (6).

\[
P(B_1 B_2) = P(B_1)+P(B_2)-P(B_1 \cap B_2)
\]

\[
= P(|A_0| \geq 1) \cap (|A_3| \geq 1) + P(|A_1| \geq 1) \cap (|A_2| \geq 1) - (1 - P(|A_1| = 0) \cup (|A_2| = 0) \cup (|A_0| = 0) \cup (|A_3| = 0))
\]

\[
= 1 - \left( \frac{1}{2} \right)^{n-2} + \left( \frac{1}{4} \right)^{n-1}
\]

(7)

then, we will calculate the \( P(B_1 B_2 | A_i = i) \) part in formula (6).

When \( A_0 = i(i \neq 0) \), we arrived at:

\[
P(B_1 B_2 | A_0 = i)
\]

\[
= P(|A_3| \geq 1) \cup (|A_1| \geq 1) \cap (|A_2| \geq 1) = 1 - P(|A_3| = 0) \cap (|A_1| = 0) \cap (|A_2| = 0) |A_0 = i)
\]

\[
= 1 - 2\left( \frac{1}{3} \right)^{n-i}
\]

(8)

Similarly, When \( A_0 = i(i = 0) \), we have:

\[
p(B_1 B_2 | A_0 = i) = 1 - 2\left( \frac{1}{2} \right)^n + \left( \frac{1}{3} \right)^n
\]

(9)
In what follows, we analyze the time and energy consumption for a tag identification.

**TABLE 4. Parameters used in simulations.**

| Parameter | Value | Parameter | Value |
|-----------|-------|-----------|-------|
| $t_1$     | 25 us | $R_{cmd}$ | 25 bit |
| $t_2$     | 25 us | $R_{bq}$  | 25 bit |
| $r$       | 160 kbps | $B_s$ | 2$^n$ bit |
| $P_{cmd}$ | 55 bit | $B_L$    | 8 bit  |
| $P_{bq}$  | 55 bit | $P_{bq}$ | 8 bit  |
| $t_{na}$  | $(R_{bq} \times 2^n) / v$ | | |
| $t_C$     | $(P_{cmd} \times 2^n \times (m-2) \times 8) / v$ | | |

Substituting formula (6)-(9) into formula (5), we can have $Sc(n) = 0.58n$. Similarly, we can have $Si(n) = 0.73n$. Furthermore, the total number of queries can be expressed as:

$$Sc(n) \approx 0.58n$$  \hspace{1cm} (10)

Similarly, we arrived at the number of collision slots $Sc(n)$ is 0.73n. Substituting (10) into (1), we can have:

$$T(t(n)) = 2 \times 0.58n + 1 = 1.16n + 1$$ \hspace{1cm} (11)

When adopting EPC-96 format to generate tag’s ID, the parameter values used in the simulation are listed in Tab. 4, the simulation results of the average number of queries required to identify one tag is plotted in Fig. 3.

**FIGURE 3.** Theoretical and simulation result of average times of reader for a tag identification.

### B. TIME CONSUMPTION

In what follows, we analyze the time and energy consumption of the SBP algorithm. According to the above analysis and formula (1), the time consumption of the SBP can be written as:

$$T(n) = \sum_{i=1}^{0.58n} (t_{B(i)} + t_{CR(i)} + 2t_1 + t_{BR} + 2t_2 + t_C) + T_{cmd}(1)$$ \hspace{1cm} (12)

where $T_{cmd}(1)$ is the transmission time of the initial CMD command. $t_{B(i)}$ is the time taken by the BQ command. $t_{BR}$ is the time spent for receiving and decoding the BQ command. $t_C$ is the transmission time of CMD command excludes the initial one. $t_{CR(i)}$ is the time spent on receiving and decoding CMD command. The values of $t_{BR}$ and $t_C$ can be found in Tab IV. The $T_{cmd}(1)$ can be expressed as:

$$T_{cmd}(1) = \frac{(P_{cmd} + 8 + 1)}{v} + t_1 + \frac{(R_{cmd} + L)}{v} + T_2$$ \hspace{1cm} (13)

The average time to identify one tag can be calculated as:

$$T = \frac{T(n)}{n}$$ \hspace{1cm} (14)

Substituting formula (12) into (14), we can have:

$$T = 1.16 \times (t_1 + t_2) + 0.58 (t_{BR} + t_C) + \frac{T_{cmd}(1)}{n}$$

$$+ \left( \sum_{i=1}^{0.58n} t_{B(i)} + \sum_{i=1}^{0.58n} t_{CR(i)} \right) \times \frac{1}{n}$$ \hspace{1cm} (15)

The part (a) in formula (15) can be derived according to Tab 2 and formula (13). Let $T_b = \sum_{i=1}^{0.58n} t_{B(i)}$, $T_{cr} = \sum_{i=1}^{0.58n} t_{CR(i)}$.

Refer to the command format in Fig. 1 we have:

$$T_b = \sum_{i=1}^{0.58n} t_{B(i)}$$

$$\leq 1 \times \left( 0.58n \times (P_{bq} + (m-2) \times 8) + \sum_{i=1}^{0.58n} |P_s| + |P_c| \right)$$ \hspace{1cm} (16)

Here in part (a) is known and part (b) is to be sought. Let $D = \sum_{i=1}^{0.58n} (|P_s| + |P_c|)$, where $|P_s| + |P_c|$ is the length of prefix contained in BQ. To find the value of D, we first assume that the traversal in the SBP algorithm is a full $M$-ary tree. In a full $M$-ary tree, the number of nodes in the kth layer is $M^k$, where $M = 2^n$. Each node in SBP contains a BQ and CMD command. Thus, D can be expressed as:

$$D = \sum_{k=1}^{N} \left( \left( \sum_{i=1}^{M^k} \frac{m k}{4} \right) + \frac{0.58n - 4^N}{N+1} \times M \right) \times P$$ \hspace{1cm} (17)

where $\sum_{i=1}^{N+1} 4^i \leq 0.58n < \sum_{i=1}^{N+1} 4^i$

In which (a) is the total length of $|P_s| + |P_c|$ in the previous $N$ layer of full $M$-ary tree, (b) is the length of $|P_s| + |P_c|$ of leaf nodes in the last layer. As we know $Sc(n) = 0.58n$ and $Si(n) = 0.73$, the probability that an $M$-ary tree is a full $M$-ary tree can be expressed as:

$$P = \frac{Sc(n) + n}{Sc(n) + n + Si(n)} = \frac{0.58n + n}{0.58n + n + 0.73n} = 0.684$$ \hspace{1cm} (18)

Substituting formula (17) and (18) into formula (16), the value of $T_b$ can be obtained. Similarly, the $T_{cr}$ can
be obtained by substituting formula (17) and (20) into formula (18).

\[
T_{cr} = \sum_{i=1}^{2^m} ICR(i) \\
= \frac{1}{\lambda} \left( 0.58n \times P_{cmd} + \sum_{i=1}^{2^m} \lambda \left( L - (|P_s| + |P_c|) \right) \right) 
\]

(19)

Average number of non-empty slots in \(2^m\) slots detected by each BQ command is written as:

\[
\lambda = \frac{Sc(n) + n}{Sc(n)} = \frac{0.58n + n}{0.58n} = \frac{1.58}{0.58} \approx 2.724 \quad (20)
\]

According to \(T_b\), \(T_{cr}\), and formula (15) the average time \(T\) for identity one tag of SBP can be derived. The derivation method of \(E(n)\) in formula (2) is very similar to \(T(n)\), the derivation process can be omitted. When adopting the data format illustrated in Fig.1 and parameters listed in Tab. 4, the simulation results of the average time and energy consumption for identifying one tag is depicted in Fig. 4.

**FIGURE 4.** Theoretical and simulation result of average time consumption and average energy consumption.

V. PERFORMANCE VALIDATION

In this section, we compare the proposed SBP with the prior \(M\)-ary-based algorithms through extensive simulations. We mainly consider the following performance metrics.

1) Communication complexity at the tag side: the number of messages that need to be transmitted to the reader when the tag is identified. Because most tags are passive or power-constrained, they require low energy consumption.

2) The number of queries: The number of interactions required by the reader to identify a tag, which affects the identification time.

3) Time consumption: The time it takes for the reader to successfully identify a tag, which is an important metric to evaluate the performance of the RFID anti-collision algorithm.

4) Energy consumption: The energy consumed by the reader to correctly identify a tag.

A. SIMULATION CONFIGURATIONS

In the simulations, the length of the tag’s ID is 96 bits, and the tag’s ID generation format is shown in Fig. 5. The format of CMD and BQ commands refer to Fig. 1. The parameters used in simulations have been listed in Tab. 4.

Where \(T_r\) means that the tag’s ID is randomly generated. \(T_h\) means that the header part is fixed and the rest part is randomly generated. \(T_m\) means that both header and manufacture parts are fixed and the rest part is randomly generated. \(T_p\) means that header, manufacture, and product are fixed, and the rest part is randomly generated. \(T_x\) means that the simulation generates 10 different types of tags, and these tags represent 1000 different types of products produced by 100 manufacturers. Before the tag identification, the reader does not know the number of tags and tag IDs.

All the results shown in Fig. 6 are obtained by averaging 100 experiments. Fig. 6 (a-d) compares the SBP algorithm with several representatives of tree-based algorithms including MQT, MCT, CwT, NAA, and STT. In Fig. 6 (a-d), all tag IDs are randomly generated with 96 bits (according to \(T_r\) in Fig. 5). Fig. 6 (e-f) compared the performance of the proposed SBP in time and energy consumption when the tag format obeys to \(T_r\), \(T_h\), \(T_m\), \(T_p\), and \(T_x\).

**FIGURE 5.** Tag’s generation format.

B. NUMBER OF TRANSMITTED MESSAGE BITS

Fig. 6(a) illustrates the average number of queries per tag identification. Observed from Fig. 6(a), the average number of queries required for various algorithms to identify a tag in order from small to large is SBP\((m = 3)\), SBP\((m = 2)\), NAA, MQT\((m = 3)\), MQT\((m = 2)\), MCT\((m = 2)\), STT, CwT, and MCT\((m = 3)\). When \(m = 3\), the average number of queries required by SBP is 0.85 which is at least 38.18% lower than NAA. When \(m = 3\), the average number of queries required by SBP is 0.85 which is at least 38.18% lower than the method NAA which has the least average paging times. When \(m = 2\), the average number of queries required by SBP is 1.15 which is at least 16.36% lower than the method NAA which has the least average paging times.

Fig. 6(b) depicts the number of transmitted bits between the reader and tags per tag identification. Observed from the results, the transmitted bits are from more to less, the ranking is STT, NAA, MQT\((m = 2)\), MCT\((m = 2)\), MQT\((m = 3)\), SBP\((m = 2)\), CwT, MCT\((m = 3)\), and the SBP\((m = 3)\), respectively. For example, when \(m = 3\), the SBP algorithm takes 144.3 bits to identify a tag, whereas MCT takes 171.5 bits. Compared with MCT which has the least average transmitted bits between the reader and tags, the SBP reduces the communication complexity by 3.6%.
C. TIME AND ENERGY CONSUMPTIONS

Fig. 6(c-d) plots time and energy consumption per tag identification. As can be seen, SBP is much better than comparative approaches in time consumption. The main reason is that the SBP algorithm uses a batch processing command to reduce the number of queries. Accordingly, the coordination time during tag identification is significantly reduced. Observed from Fig. 6(c), the time consumption is from less to more, the ranking is SBP, MQT, MCT, NAA, STT, and CwT, the proposed SBP reduces the time consumption by 22.94%, respectively, compared to MQT which has the least time-consuming. The energy consumption of the algorithm is proportional to the time consumption of the reader and tag. The SBP algorithm proposed in this paper is superior to the comparison of several algorithms in terms of reader time consumption and tag time consumption, so the energy consumption of the SBP algorithm is also lower than those algorithms that are compared in this paper. Observed from Fig. 6(d), the proposed SBP reduces energy consumption by 22.28% respectively, compared to MQT which has the least energy consumption.

D. EFFECT OF TAG DISTRIBUTION ON SBP ALGORITHM

Fig. 6(e-f) compare the time and energy consumption per tag identification with various ID distributions. It can be seen from Figure (e) that when the tags are generated in different distribution, the time consumption of the SBP algorithm is $T_p, T_m, T_h, T_r$, and $T_x$ from less to more. The main reason is that the parameter pre (the prefix of tags) carried in the CMD command only needs to be sent once, and all tags whose register regM value is not equal to null will return the remaining ID to the reader in the corresponding time slot. Therefore, the lower average highest collision bit detected by the reader will lead to the longer prefix and the shorter remaining ID, which in turn leads to the smaller amount of interactive data between reader and tags, the smaller SBP algorithm’s time overhead. Observed from Fig. 6(e-f), under different ID distributions, the proposed SBP algorithm only needs a maximum of 1.48ms and 1.35mJ to recognize a label, which is better than the benchmark algorithms compared in this paper in time and energy consumption.

In summary, simulation results in Fig. 6(a-d) verify that the proposed SBP is superior to other tree-based algorithms under the condition that the tag’s ID is randomly distributed. Simulation results in Fig. 6(e-f) verify that regardless of tag ID distribution, the proposed SBP algorithm can maintain the constant performance.

E. EFFECT OF PATH LOSS AND CAPTURE EFFECT ON SBP ALGORITHM

Due to the path loss and capture effect in communication links, there will be a certain discrepancy between the actual performance and the ideal performance of an algorithm. In what follows, we will explore the actual performance of the SBP algorithm. Firstly, we analyze the influence of path loss and the capture effect on identification performance.

The reason for the path loss is mainly due to the distance difference between the reader and tag, obstacles, transmission power, and other factors. As a result, the transmitted and received signal is attenuated, causing the reader to identify the tag in vain. When path loss occurs, the collision slot may be erroneously detected as a success slot or an empty slot,
and the success slot may be erroneously detected as an empty slot, which is shown in Fig. 7. In tree-based algorithms, the involved tags resulting from path loss cannot be identified in the current identification round. And they will participate in the next identification round.

Since the distance between the tags and the reader is different, the backscatter signal strength at the reader side is different. When the signal strength of a certain tag is significantly stronger than other tags, it will cause a capture effect. The capture effect will only affect the collision slot. When it occurs, the original collision slot may be converted into a success slot, thereby speeding up the identification process.

In the following, we evaluate the performance of the algorithm in a realistic environment (with path loss and capture effects). In simulations, the probability that the tag can be successfully detected by the reader is defined as $P_s$, where $P_s$ is from 0.9 to 1. The occurrence probability of capture effect is defined as $P_c$, where $P_c$ is from 0 to 0.3. The tag format can be referred to $T_r$ in Fig. 5.

As can be seen from Fig. 8, the smaller the $P_s$ value, the fewer number of tags per second that the algorithm identifies. The larger the $P_c$ value, the more tags per second the algorithm identifies. In summary, when $P_s$ and $P_c$ change dynamically, the average number of tags per second identified by the SBP algorithm is the highest, followed by the MQT algorithm, and the MCT algorithm is the least.

![FIGURE 7. Slot transform relationship.](image1)

![FIGURE 8. Average number of tags identification by the reader per second under practical environment.](image2)

![FIGURE 9. Average energy for a tag identification under practical environment.](image3)

Fig. 9 shows the average energy consumption per tag identification in a realistic environment. It can be seen that the energy consumption of identifying a tag decreases as the value of $P_c$ increases, and the larger the value of $P_s$, the smaller the energy consumption. As observed in Fig. 9, the overall energy consumption of SBP is lower than that of MQT and MCT algorithms.

VI. CONCLUSION

In this paper, we propose a smart batch processing (SBP) protocol for tag identification. In the SBP, once the reader detects a tag collision, it will first form an optimal $M$-ary traversal tree and then send a batch processing command to allow multiple tags to be identified in a slot. Through theoretical analysis and simulations, we prove that our proposed SBP algorithm is superior to other tree-based algorithms in terms of the number of queries, communication complexity, identification times, and reader energy consumption. Besides, the performance of SBP is verified under different tag ID distributions. Simulation results show that the proposed SBP still maintains a good performance under various tag ID distributions. Finally, we evaluate the performance of the SBP algorithm in a realistic environment with path loss and capture effects, which also proves that it is still better than other comparative algorithms.

REFERENCES

[1] P. Solic, Z. Blazevic, M. Skiljo, L. Patrono, R. Colella, and A. J. J. P. C. Rodrigues, “Gen2 RFID as IoT enabler: Characterization and performance improvement,” IEEE Wireless Commun., vol. 24, no. 3, pp. 33–39, Jun. 2017.
[2] J. Su, Z. Sheng, V. C. M. Leung, and Y. Chen, “Energy efficient tag identification algorithms for RFID: Survey, motivation and new design,” IEEE Wireless Commun., vol. 26, no. 3, pp. 118–124, Jun. 2019.
[3] L. Arjona, E. Onieva, A. Perallos, and H. Landaluce, “Fast fuzzy anti-collision protocol for the RFID standard EPC Gen-2,” Electron. Lett., vol. 52, no. 8, pp. 663–665, Apr. 2016.
[4] L. Barletta, F. Borgonovo, and M. Cesana, “A formal proof of the optimal frame setting for dynamic-frame aloha with known population size,” IEEE Trans. Inf. Theory, vol. 60, no. 11, pp. 7221–7230, Nov. 2014.
[5] W.-T. Chen, “Optimal frame length analysis and an efficient anti-collision algorithm with early adjustment of frame length for RFID systems,” IEEE Trans. Veh. Technol., vol. 65, no. 5, pp. 3342–3348, May 2016.
[6] J. Su, Z. Sheng, L. Xie, G. Li, and A. X. Liu, “Fast splitting-based tag identification algorithms for anti-collision in UHF RFID system,” IEEE Trans. Commun., vol. 67, no. 3, pp. 2527–2538, Mar. 2019.
[7] L. Zhang, W. Xiang, and X. Tang, “An adaptive anti-collision protocol for large-scale RFID tag identification,” IEEE Wireless Commun. Lett., vol. 3, no. 6, pp. 601–604, Dec. 2014.

[8] Y.-C. Lai, L.-Y. Hsiao, H.-J. Chen, C.-N. Lai, and J.-W. Lin, “A novel query tree protocol with bit tracking in RFID tag identification,” IEEE Trans. Mobile Comput., vol. 12, no. 10, pp. 2063–2075, Oct. 2013.

[9] J. Su, Z. Sheng, A. X. Liu, Y. Han, and Y. Chen, “A group-based binary splitting algorithm for UHF RFID anti-collision systems,” IEEE Trans. Commun., vol. 68, no. 2, pp. 998–1012, Feb. 2020.

[10] J. Su, Z. Sheng, D. Hong, and G. Wen, “An effective frame breaking policy for dynamic framed slotted aloha in RFID,” IEEE Commun. Lett., vol. 20, no. 4, pp. 692–695, Apr. 2016.

[11] A. Abbasian and M. Safkhani, “CNCAA: A new anti-collision algorithm using both collided and non-collided parts of information,” Comput. Netw., vol. 172, May 2020, Art. no. 107159, doi: 10.1016/j.comnet.2020.107159.

[12] N. Bagheri, P. Alenaby, and M. Safkhani, “A new anti-collision protocol based on information of collided tags in RFID systems,” Int. J. Commun. Syst., vol. 30, no. 3, p. e2973, Feb. 2017.

[13] H. Landaluce, A. Perallos, E. Onieva, L. Arjona, and L. Bengtsson, “An energy and identification time decreasing procedure for memoryless RFID tag anticollision protocols,” IEEE Trans. Wireless Commun., vol. 15, no. 6, pp. 4234–4247, Jun. 2016.

[14] L. Zhang, W. Xiang, X. Tang, Q. Li, and Q. Yan, “A time- and energy-aware collision tree protocol for efficient large-scale RFID tag identification,” IEEE Trans. Ind. Informat., vol. 14, no. 6, pp. 2406–2417, Jun. 2018.

[15] X. Jia, Q. Feng, and C. Ma, “An efficient anti-collision protocol for RFID tag identification,” IEEE Commun. Lett., vol. 14, no. 11, pp. 1014–1016, Nov. 2010.

[16] D.-G. Zhang, G. Li, Z.-H. Pan, and Y.-P. Liang, “A new anti-collision algorithm for RFID tag,” Int. J. Commun. Syst., vol. 27, no. 11, pp. 3312–3322, 2014, doi: 10.1002/dac.2543.

[17] L. Zhang and W. Xiang, “An energy- and time-efficient M-ary detecting tree protocol,” in Proc. IEEE Int. Conf. Commun. (ICC), London, U.K., Jun. 2015, pp. 2882–2887.

[18] Y. Kim, S. Kim, S. Lee, and K. Ahn, “Improved 4-ary query tree algorithm for anti-collision in RFID system,” in Proc. Int. Conf. Adv. Inf. Netw. Appl., 2009, pp. 699–704.

[19] J. Shin, B. Jeon, and D. Yang, “Multiple RFID tags identification with M-ary query tree scheme,” IEEE Commun. Lett., vol. 17, no. 3, pp. 604–607, Mar. 2013.

[20] L. Zhang, J. Zhang, and X. Tang, “Assigning tree slotted aloha RFID tag anti-collision protocols,” IEEE Trans. Wireless Commun., vol. 12, no. 11, pp. 5493–5504, Nov. 2013.

[21] J. Su, Z. Sheng, G. Wen, and V. C. M. Leung, “A time efficient tag identification algorithm using dual prefix probe scheme (DPPS),” IEEE Signal Process. Lett., vol. 23, no. 3, pp. 386–389, Mar. 2016.

[22] M. Shahzad and A. X. Liu, “Probabilistic optimal tree hopping for RFID identification,” IEEE/ACM Trans. Netw., vol. 23, no. 3, pp. 796–809, Jun. 2015.

[23] L. Pan and H. Wu, “Smart trend-traversal protocol for RFID tag arbitration,” IEEE Trans. Wireless Commun., vol. 10, no. 11, pp. 3565–3569, Nov. 2011.

[24] Y. Zhu, W. Jiang, Q. Zhang, and H. Guan, “Energy-efficient identification in large-scale RFID systems with handheld reader,” IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 5, pp. 1211–1222, May 2014.