# Fast RFID Polling Protocols

Jia Liu[†‡], Bin Xiao[‡], Xuan Liu[§†] and Lijun Chen[†]
[†]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[‡]*Department of Computing, The Hong Kong Polytechnic University, China*
[§]*College of Computer Science and Electronic Engineering, Hunan University, China*
*Email: liujia@smail.nju.edu.cn, csbxiao@comp.polyu.edu.hk, xuan_liu@hnu.edu.cn, chenlj@nju.edu.cn*

***Abstract*—Polling is a widely used anti-collision protocol that interrogates RFID tags in a request-response way. In conventional polling, the reader needs to broadcast 96-bit tag IDs to separate each tag from others, leading to long interrogation delay. This paper takes the first step to design fast polling protocols by shortening the polling vector. We first propose an efficient Hash Polling Protocol (HPP) that uses hash indices rather than tag IDs as the polling vector to query each tag. The length of the polling vector is dropped from 96 bits to no more than $\lceil log(n) \rceil$ bits ($n$ is the number of tags). We then enhance HPP (EHPP) to make it not only more efficient but also more steady with respect to the number of tags. To avoid redundant transmissions in both HPP and EHPP, we finally propose a Tree-based Polling Protocol (TPP) that reserves the invariant portion of the polling vector while updates only the discrepancy by constructing and broadcasting a polling tree. Theoretical analysis shows that the average length of the polling vector in TPP levels off at only 3.44, 28 times less than 96-bit tag IDs. We also apply our protocols to collect tag information and simulation results demonstrate that our best protocol TPP outperforms the state-of-the-art information collection protocol.**

***Keywords*-RFID; polling; information collection**

## I. Introduction

Radio Frequency IDentification (RFID) is a popular non-contact technology that exploits radio-frequency electromagnetic fields to transfer data. With the rock-bottom price and rapid development of manufactural technologies, RFID has been widely employed in various areas, such as object tracking [1], [2], [3], supply chain management [4], [5], [6], warehouse inventory [7], [8], [9], [10]. Considering the proliferation of RFID tags, time-efficient anti-collision protocols are essential to successfully collect information from tagged objects.

Polling, as a common anti-collision protocol, allows us to interrogate RFID tags in a request-response way. It serves a broad range of purposes. For example, polling can be used to detect or identify missing tags [11], [12], [13], [14] with 1-bit response from a tag showing its presence against theft. Polling can be applied to collect tag information for monitoring the status of tags and products in sensor-augmented RFID networks [15], [16], such as the energy level of batteries, the temperature of chilled food, or product information stored in the tag's memory.

The remarkable advantage of polling is that the interrogating request and response are a one-to-one mapping, such that only one tag is interrogated at a time, completely avoiding tag-to-tag collisions in the open wireless channel. This feature makes polling contain only useful singleton slots (exact one tag replies each time), instead of useless empty slots (none of tags replies) and collision slots (two or more tags reply concurrently) that happen in ALOHA-based protocols [12], [8], [17], [18]. In the Conventional Polling Protocol (CPP), however, the reader needs to transmit tedious tag IDs to separate each tag from others, leading to long polling delay. We refer to the broadcasting bits (from a reader) used for exclusively polling a tag as *polling vector* in the sequel. Clearly, the polling vector of CPP is the 96-bit tag ID.

The existing work of polling design is to reduce the number of polling phases [12], [19], each of which, however, still adopts inefficient CPP. With the proliferation of RFID tags, this design is likely to take a long time to finish the polling task even for a medium-size RFID system. The most related work is the Coded Polling (CP) protocol that shortens the length of the polling vector by half through validating the cyclic redundancy code [19]. However, CP is far away from a time-efficient polling protocol as the 48-bit polling vector is still too long to quickly interrogate tags.

This paper takes the first attempt to exploit fast polling protocols from the view of shortening the polling vector. Our major contribution is that we make a fundamental improvement on the polling performance by taking a different design path. A series of protocols are proposed to progressively achieve this goal. The key idea behind our protocols is to minimize the polling vector as well as to totally avoid slot waste. We first present an efficient Hash Polling Protocol (HPP) that uses a new, short hashed index rather than the tedious tag ID as the polling vector to exclusively query each tag. Theoretical analysis shows that the polling vector of HPP yields the upper bound of $\lceil \log_2 n \rceil$ bits, where $n$ is the number of tags. Although HPP greatly decreases the polling overhead compared with CPP (96 bits), the length of the polling vector increases with $n$. We thus design an Enhanced HPP (EHPP) that divides a large tag set into several small subsets and respectively interrogates tags in each of them. EHPP not only improves the polling efficiency but also keeps the length of the polling vector steady, regardless

of the number of tags. To avoid redundant information transmissions in HPP and EHPP, we finally propose a Tree-based Polling Protocol (TPP) that constructs and broadcasts a polling tree to further improve the polling performance. With such a polling tree, TPP transmits only differential bits between the current polling vector and the previous one, lessening information redundant. Theoretical analysis shows that the upper bound of TPP's polling vector levels off at only 3.44 bits, 28 times less than 96-bit IDs.

We conduct extensive simulations to evaluate the performance gain of our protocols, in terms of the length of the polling vector and time efficiency. Simulation results demonstrate that our best protocol TPP shortens the polling vector from 96 bits to only about 3 bits, which yields 31 times less bits than CPP. Besides, compared with the state-of-the-art ALOHA-based information collection protocol MIC [15], TPP not only reduces the inventory time by 14.8% when collecting 1-bit tag information (showing a tag's presence), but also requires less storage space at the tag side due to less hash functions required by tags.

The rest of this paper is organized as follows. Section II defines the problem. Section III presents the hash polling protocol and its enhanced version. Section IV proposes the tree-based polling protocol. Section V evaluates our protocols. Section VI presents related work. Finally, Section VII concludes this paper.

## II. Problem Definition

### A. System Model

An RFID system generally consists of a backend server, multiple readers, and a large number of tags. Each tag with a unique tag ID can communicate with a reader directly, but the tags cannot communicate amongst themselves. All readers are connected to the backend server that can provide high computing power and large data storage. We can logically treat these readers as one if they are well synchronized and scheduled. To simplify the description, our protocols are presented for a single reader, but they can be easily modified for multiple readers when the collision-free transmission schedule among the readers is established. We assume that the reader has the knowledge of all tag IDs in advance. It is a fundamental assumption for many system-level applications, such as missing-tag identification [11], [12], [20], information collection [15], [19], [16]. In this paper, the communication between the reader and tags follows the *Reader Talks First* mode according to the C1G2 standard [21]. Namely, every tag waits for the reader's command before responding.

### B. Conventional Polling Protocol

The Conventional Polling Protocol (CPP) is an intuitive polling solution that serves as our baseline protocol for comparisons. In CPP, the reader first broadcasts a tag ID and then waits for its reply. All tags in the interrogation zone keep listening and only the tag whose ID exactly matches the broadcast one replies to the reader. In this way, only one tag is interrogated at a time, effectively avoiding tag-to-tag collisions in the open wireless channel. CPP, however, cannot meet real-time RFID-enabled applications since transmitting 96-bit tag IDs is time-consuming. Fig. 1 depicts the execution time with respect to the length of the polling vector when collecting 1-bit information from a tag (the parameter setting follows the C1G2 standard [21], see Section V-A). Clearly, the execution time is proportional to the length of the polling vector. Our objective in this paper is to shorten the polling vector and thereby reduce the polling time.

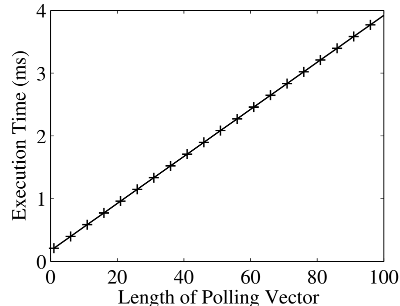

Figure 1. Execution time with respect to the length of the polling vector.

In some practical cases, tags may share a common prefix of their IDs. For example, tags affixed to the same class of items have the identical category ID. The reader herein can achieve better polling performance in a more sophisticated way: 1) broadcast the category ID to mask a tag subset; 2) truncate and transmit the left, differential bits to poll each tag in this subset. Such enhanced CPP improves the polling performance but relies on the specific distribution of tag IDs. Additionally, even when all tags share the same prefix, e.g., 32-bit category ID, it still needs above 64-bit polling vectors to poll each tag, far away from efficient polling. In this paper, we consider a more general case without any assumption on the distribution of tag IDs.

### C. Problem Definition

Consider a large RFID system with a tag set $T = \{t_1, t_2, t_3, ..., t_n\}$ representing all $n$ RFID tags in the interrogation zone. The fast polling problem studied in this paper is to collect $m$-bit information from each tag $t_i$ in a request-response way as quickly as possible, where $m \geq 1$ and $1 \leq i \leq n$. CPP provides an intuitive polling scheme but suffers from dreary long-ID transmission. In this paper, we therefore target at time-efficient polling protocols that not only take full use of every time slot but also minimize the length of the polling vector.

## III. Hash Polling Protocol

In this section, we first propose the Hash Polling Protocol (HPP) that avoids transmitting long tag IDs. We then

enhance HPP (EHPP) to ensure stable polling performance regardless of the number of tags.

### A. HPP Overview

The key idea behind HPP is to exclusively poll tags by assigning each of them a new, short hashed index that replaces long tag IDs. To take full advantage of every time slot, HPP uses only the index picked by a single tag to poll such a tag. HPP generally consists of multiple inventory rounds, in each of which about 36.8%∼60.7% of tags are read. Other unread tags will participate and be interrogated in following query rounds. Details are given below.

### B. HPP Description

Consider an arbitrary inventory round. Assume that there are $n'$ unread tags before this round. Clearly, $n' = n$ in the first round. The reader first initiates this query round by broadcasting a specific request with the parameters $\langle h,r \rangle$, where $h$ is the index length satisfying $2^{h-1} < n' \leq 2^h$ and $r$ is a random seed. Upon receiving the request, each tag picks an index $\mathcal{H}(r, id) \mod 2^h$, where $id$ is the tag ID and $\mathcal{H}(\cdot)$ is a hash function. If the index is less than $h$ bits, pad zeros in front of it. We refer to the indices picked by no tag, exactly one tag, and more than one tag as *empty index*, *singleton index*, and *collision index*, respectively.

Since the reader has access to all tag IDs, it can pre-compute which index each tag picks. The reader then sifts out all singleton indices in this round and broadcasts them in sequence. All tags keep listening and only the tag that picks the broadcast index replies to the reader. Once a tag is interrogated, it will go to sleep in the following protocol execution. The current round terminates after the reader transmits all singleton indices. All left tags picking the collision indices keep active and will participate in the next round query. The inventory procedure repeats round by round until all tags are interrogated.

Note that only singleton indices can be used to poll tags. That is because each of them is picked by exactly one tag, which ensures an exclusive response each time, instead of the useless no responses or collision responses due to transmitting empty indices or collision indices.

Fig. 2 illustrates the polling process of HPP. The reader initiates a new round with parameter $h = 2$ and the tags $A$, $B$, $C$, and $D$ randomly pick an index. Clearly, '10' is an empty index, '01' is a collision index, '00' and '11' are singleton indices. The reader first broadcasts '00'; $C$ picking this index replies to the reader. The reader then transmits the other singleton index '11'; $B$ responds to the reader. Because there are no singleton indices any more, the current round terminates. The tags $B$ and $C$ now keep silent, whereas $A$ and $D$ stay alert for participating in the following inventory rounds.

Be aware of the difference between HPP and ALOHA-based approaches. In the ALOHA-based protocols, each tag individually chooses a slot and the reader has to in turn traverse every slot (from the first one to the last); the useless empty slots and collision slots thus bring unnecessary communication overhead. HPP, instead, directly broadcasts singleton indices and skips over empty indices and collision indices to poll tags. Therefore, the number of polling is exactly equal to the number of singleton indices in a round. From a global view of HPP, the total number of polling is the same with the number of tags, completely avoiding slot waste.

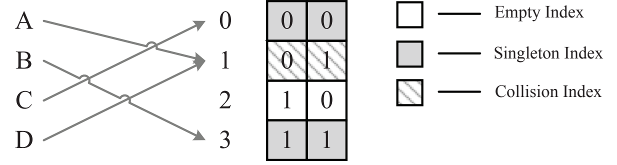


Figure 2. The polling process of HPP. The index length $h = 2$. Four tags $A$, $B$, $C$, and $D$ randomly pick an index that is in [0, 3]. Only the singleton indices, i.e., '00' and '11', can be used to poll $C$ and $B$, respectively.

### C. HPP's Polling Vector

To clarify the polling overhead, we analyze the expected length of the polling vector in HPP. Take the $i$th, $i \geq 1$, round into account. Suppose there are $n_i$ unread tags before this round and the index length is $h_i$ that satisfies $2^{h_i - 1} < n_i \leq 2^{h_i}$. The probability that an index is a singleton index is

$$p_i = \binom{n_i}{1}(\frac{1}{f_i})(1 - \frac{1}{f_i})^{n_i - 1} \approx \frac{n_i}{f_i} \times e^{-\frac{n_i - 1}{f_i}}, \quad (1)$$

where $f_i = 2^{h_i}$. Since a singleton index corresponds to a single tag to be interrogated, 36.8%∼60.7% of unread tags will participate in replying according to (1). Let $n_{si}$ be the number of singleton indices in the $i$th round, we have

$$n_{si} = f_i \times p_i = n_i \times e^{-\frac{n_i - 1}{f_i}}. \quad (2)$$

After the $i$th query round, the number of remaining unread tags before the $(i+1)$th round is

$$n_{i+1} = n_i - n_{si} = n_i \times (1 - e^{-\frac{n_i - 1}{f}}), \quad (3)$$

where $i \geq 1$ and $n_1 = n$ ($n$ is the total number of tags). We then have the average length $w$ of the polling vector:

$$w = \frac{\sum_{i=1}^{k} h_i \times n_{si}}{n} = \frac{\sum_{i=1}^{k} h_i \times n_i \times e^{-\frac{n_i - 1}{f_i}}}{n}, \quad (4)$$

where $k \geq 1$ satisfying $n_k \neq 0$ and $n_{k+1} = 0$. Once given $n$, we can derive the $w$ according to (4). Here, we give a rough upper bound $w^+$ of $w$:

$$w^+ = \lceil \log_2 n \rceil. \quad (5)$$

Consider the $i$th inventory round. The length of the polling vector is actually equal to the index length $h_i$, which satisfies $2^{h_i - 1} < n_i \leq 2^{h_i}$. Since $n_i \leq n_1 = n$, we have $h_i \leq \lceil \log_2 n \rceil$, $i \geq 1$. Fig. 3 shows the average length $w$ of the polling

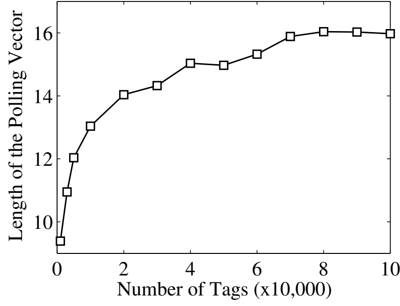

Figure 3. The average length $w$ of the polling vector with respect to the number $n$ of tags.

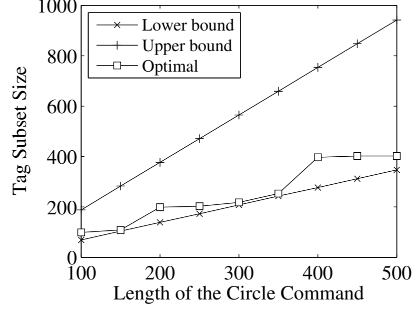

Figure 4. The optimal subset size $n'$ with respect to the length $l_c$ of the circle command.

vector with respect to the number $n$ of tags according to (4). We observe that all polling vectors are under 16 bits, which are much less than 96-bit IDs in CPP. However, $w$ almost monotonously increases with $n$. For example, when $n = 1000$, $w$ approximates 10, whereas $w$ is around 16 when $n = 10^5$. This gives us huge room for further improvement, motivating us to design a more scalable protocol that is insensitive to the number of tags.

### D. Enhanced HPP

To shorten the polling vector under large tag population, EHPP divides the entire tag set into many small subsets and interrogates each of them respectively using HPP. The period for querying a tag subset is defined as a *circle*. EHPP is thus likely to consist of multiple circles for polling the entire tag set. With optimal parameter settings, EHPP can not only improve HPP's polling efficiency but also keep the polling vector steady regardless of the number of tags.

One key challenge of EHPP is how to separate a tag subset with the expected number of tags from all tags as required. There are generally two ways: 1) Bit mask: In the C1G2 standard, a *Select* command limits the number of participated tags using a bit mask, and only the tags whose IDs (or some bits of the ID) exactly match this mask will be active in the current circle [22]. Although a bit mask gives a fine-grained tag filtering, it is unavailable to sift out the specified number of tags under an arbitrary tag ID distribution. We thus resort to the other way 2) Probability: If a reader covering $n$ tags broadcasts a query command and each tag responds with probability $p$, we can expect $p \times n$ tag responses [23]. However, the reader is unable to know which tags are chosen since each tag randomly participates in the interrogation. In light of this, we propose a variant of this probability strategy: Before the polling process, a reader first broadcasts a query command to initiate a circle with parameters $\langle f, F, r \rangle$ and each tag individually chooses an index ranging from 0 to $F$ by calculating $\mathcal{H}(r, ID) \mod F$. Only the tags picking the index no more than $f$ will join the query in the current circle. To this point, $\frac{f}{F} \times n$ tags will be active. By dynamically adjusting $f$ and $F$, we can get the expected number of active tags as required.

The other challenge of EHPP is how to set the number of tags in each subset to minimize overall polling overhead. Considering a large tag set, we evenly split it into several small subsets. Since each tag subset has the same number of tags, the polling efficiency in each circle is identical. Assume the number (which is also called the subset size) of tags in each subset is $n'$. The problem is thereby reduced to minimize the length $w'$ of the polling vector in a circle by optimizing $n'$.

**Theorem 1.** *Given the length $l_c$ of the circle command, the optimal subset size $n'$ must be the interval [$l_c \ln 2$, $e l_c \ln 2$], where $e$ is the natural constant.*

*Proof:* Given $l_c$ and $n'$, we have the length $w'$ of the polling vector in this circle:

$$w' = \frac{h(n') + l_c}{n'} = \frac{h(n')}{n'} + \frac{l_c}{n'},$$

where $h(n')$ is the number of bits used to poll $n'$ tags and $\frac{h(n')}{n'}$ is the average length of the polling vector in HPP. As aforementioned, HPP consists of several execution rounds. If we just consider the first round, there are more than $\frac{n'}{e}$ tags are polled by $(\log_2 n')$-bit polling vectors. For another, all polling vectors in this circle are less than $\log_2 n'$ bits. Therefore, we have:

$$\frac{1}{e} \log_2 n' \leq \frac{h(n')}{n'} \leq log_2 n'.$$

Let $\frac{h(n')}{n'} = \mu \times log_2 n'$, where $\frac{1}{e} \leq \mu \leq 1$. We have:

$$w' = \mu \times log_2 n' + \frac{l_c}{n'}.$$

By deriving $\frac{dw'}{dn'} = 0$, we get the minimal $w'$ when $n' = \frac{l_c \ln 2}{\mu}$, $\frac{1}{e} \leq \mu \leq 1$. Therefore, we have the interval in which the optimal $n'$ lies. ■

According Theorem 1, we can numerically search the optimal $n'$ for an arbitrary given $l_c$ to maximize the polling efficiency. Fig. 4 depicts the optimal subset size $n'$ with respect to the length $l_c$ of the circle command. Clearly, the bigger $l_c$ is, the bigger $n'$ is. That is because the longer circle command causes more communication overhead in

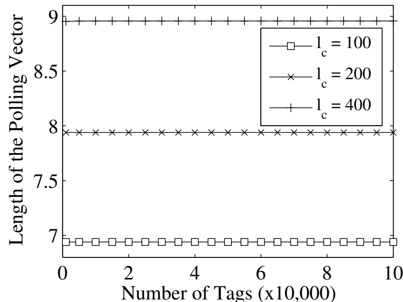


Figure 5. The average length $w$ of the polling vector with respect to the number $n$ of tags in EHPP.

each circle, promoting EHPP to reducing the number of circles by increasing the subset size. Besides, the optimal $n'$ is sandwiched between the upper bound and the lower bound, which well validates Theorem 1. Note that, as HPP will be invoked many times during the execution of EHPP, the communication overhead to initiate each round of HPP may not be ignored. We count this overhead in the simulation of EHPP (see Section V).

Fig. 5 illustrates the polling efficiency with respect to the number of tags. In this figure, we consider three kinds of circle commands with the lengths 100, 200, and 400, respectively. Given the length $l_c$, we look up the corresponding subset size according to Fig. 4. We observe that EHPP dramatically reduces the length of the polling vector compared with HPP. For instance, to query a set of $10^5$ tags, EHPP needs only about 7.94 bits to poll each tag when $l_c = 200$, whereas HPP consumes about 15 bits. Additionally, when $l_c$ is fixed, the average length $w$ of the polling vector almost remains stable, regardless of the number of tags. Although with these advantages, we note that EHPP's polling vector increases with $l_c$ due to more communication overhead caused by broadcasting longer circle commands. To break through this limitation, we are supposed to explore a more stable and efficient polling protocol.

## IV. Tree-Based Polling Protocol

In this section, we propose a Tree-based Polling Protocol (TPP) that further improves the polling efficiency by avoiding redundant transmissions of polling vectors in HPP and EHPP.

### A. Motivation of TPP

In both HPP and EHPP, the common prefixes of singleton indices are repeatedly broadcast. For example, consider two singleton indices '000' and '001' picked by tags $A$ and $B$. The reader has to broadcast '000' and '001' to poll $A$ and $B$, respectively. The common prefix '00' is thus broadcast twice, causing unnecessary communication overhead. Hence, once upon removing this redundant, we can further shorten the length of the polling vector and improve the polling efficiency.

### B. TPP Overview

Similar to HPP, TPP also consists of multiple query rounds. The reader in TPP, however, does not directly broadcast the singleton indices in each round. Instead, TPP first constructs a binary polling tree based on all singleton indices and then polls the corresponding tags using such a tree. Thanks to the polling tree, all common prefixes are transmitted only once, saving the communication overhead. Details are given below.

### C. TPP Description

Consider an arbitrary inventory round. TPP consists of three phases. *1) Picking index*: the reader initializes this round and each tag randomly picks an index. *2) Building polling tree*: the reader constructs a binary polling tree based on all singleton indices instead of directly broadcasting them. *3) Tree-based polling*: the reader interrogates corresponding tags by broadcasting the binary polling tree.

*1) Picking Index:* In TPP, the reader initiates a new query round by sending an interrogation request with parameters $\langle h,r \rangle$, where $h$ is the index length determined by the number of unread tags, $r$ is a random seed. Upon receipt of the query, each tag individually picks an index $\mathcal{H}(r, id) \mod 2^h$. If the index is less than $h$ bits, pad zeros in front of it. Meanwhile, the reader pre-computes all picked indices and sifts out only singleton indices. In this phase, be aware of the difference of $h$ between TPP and HPP. We will analyze the optimal setting of $h$ in Section IV-D.

*2) Building Polling Tree:* In this phase, the reader in TPP builds a binary tree based on singleton indices rather than directly broadcasting them. This binary tree will be leveraged to poll corresponding tags in the next phase; we refer to it as *binary polling tree* (*polling tree* for short). To build the polling tree, we create a virtual node as the root, traverse all singleton indices one after another, and insert new node into the tree based on these singleton indices. Consider an arbitrary $h$-bit singleton index $\mathbb{S}$. We scan each bit of $\mathbb{S}$ in sequence and simultaneously use a pointer to record the current position in the tree. Initially, the pointer stays at the root node and $\mathbb{S}$ is scanned from the first bit. If the current bit is 0, we create a left child of the node pointed by the pointer. Otherwise, a right child will be created. After that, we move the pointer to the new created node and check $\mathbb{S}$'s next bit. The above steps repeat. After all $h$ bits are inserted into the tree, we reset the pointer to the root and insert the next singleton index. Note that if there has been a left/right child, we do nothing but update the position of the pointer. Fig. 6 illustrates the construction of a polling tree when inserting five singleton indices ($h$=3) picked by tags $A$, $B$, $C$, $D$, and $E$. As we can see, the root of the tree is a virtual node; the left child denotes the bit '0' and the right child denotes the bit '1'.

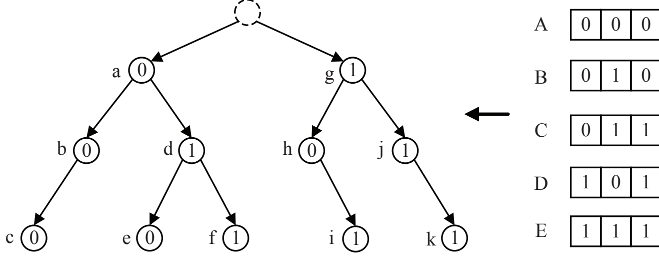


Figure 6. The construction of the binary polling tree. The root is a virtual node. The left child denotes the bit 0 and the right child denotes the bit 1.

*3) Tree-based Polling:* In this phase, we show how to poll the corresponding tags with only the polling tree. A leaf node in the polling tree corresponds to a singleton index, as the consecutive bits from the root to the leaf node make up a singleton index. For example, $a \rightarrow b \rightarrow c$ indicates '000' in Fig. 6. The leaf nodes with common ancestor nodes have the common prefix, e.g., $e$ and $f$ have the common prefix '01' since they share common ancestor nodes $a$ and $d$. This phase aims to broadcast all singleton indices stored in the polling tree without repeatedly transmitting common prefixes.

Given a polling tree with $n'$ leaf nodes. We first execute the pre-order traversal (one kind of depth-first traversal) of this tree and record the trace of the traversal, denoted by $Seq = \{b_0, a_1, a_2, ..., a_{h-1}, b_1, a_h, ..., b_2, ..., b_{n'}\}$, where $b_0$ is the root node, $b_j$ $(1 \leq j \leq n')$ is the leaf node, and $a_i$ $(i \geq 1)$ is the non-leaf node except the root. Let $Seq[j]$ be the nodes between $b_{j-1}$ and $b_j$, including $b_j$ but excluding $b_{j-1}$. For example, $Seq[1]$ is $\{a_1, a_2, ..., a_{h-1}, b_1\}$. The reader then broadcasts $Seq[j]$ in turn and waits for a tag's reply. Every tag hears the RF channel and holds an $h$-bit array at the same time, denoted by $\mathbb{A}$. Assume that the length of $Seq[j]$ is $k$, $1 \leq k \leq h$. Upon receipt of $Seq[j]$, each tag updates the last $k$ bits of its $\mathbb{A}$ with $Seq[j]$. Clearly, both the first $(h-k)$ bits in $\mathbb{A}$ and $Seq[j]$ constitute the $j$th singleton index; the unique tag picking this slot is supposed to respond to the reader. The above process repeats until all $n'$ singleton indices are broadcast.

Fig. 7 shows an example of the polling process based on the polling tree in Fig. 6. The trace of the pre-order traversal of this polling tree is $Seq = \{ab\mathbf{c}d\mathbf{ef}gh\mathbf{i}j\mathbf{k}\}$ (the bold nodes are leaf nodes). There are five singleton indices selected by five tags $A$, $B$, $C$, $D$, and $E$. The index length $h$ is equal to 3. (a) The reader first broadcasts the singleton index '000' represented by $Seq[1] = \{ab\mathbf{c}\}$. Upon receiving this bit array, the tags $A$, $B$, $C$, $D$, and $E$ update their $\mathbb{A}$s with '000'. Because $A$'s index is equal to its $\mathbb{A}$, $A$ replies to the reader. (b) $Seq[2]$ is $\{d\mathbf{e}\}$, the reader thus broadcasts '10'. The remaining unread tags $B$, $C$, $D$, and $E$ update the last two bits of $\mathbb{A}$s with '10'. $B$'s index matches the current $\mathbb{A}$ and it replies to the reader. (c) $Seq[3]$ is $\{\mathbf{f}\}$; the reader broadcasts '1'. $C$, $D$, and $E$ update the last bit of $\mathbb{A}$s with '1'. $C$ is polled. (d) $Seq[4] = \{gh\mathbf{i}\}$; the reader broadcasts '101'. $D$ and $E$ update their $\mathbb{A}$s with it. $D$ is polled. (e) $Seq[5]$ is $\{j\mathbf{k}\}$; the reader broadcasts '11'. $E$ updates the last two bits of its $\mathbb{A}$ and $E$ is polled. Instead of sending $h \times 5 = 15$ bits, the reader in this round transmits only 11 bits in total, greatly reducing communication overhead.

Although the reader broadcasts the polling tree only once, it is equivalent to broadcasting all singleton indices to poll corresponding tags in a single round. The main reason behind this is that every tag holds a bit array $\mathbb{A}$ and updates only different bits between the current singleton index and the previous one each time. The common prefix information is thus reserved. Whit the proliferation of tags, more common prefixes will be generated, producing more performance gain.

### *D. TPP's Polling Vector*

In this subsection, we analyze the average length of the polling vector in TPP. Consider the polling tree in the $i$th query round. Assume that the height of the tree is $h_i$ (the length of the single index) and the number of leaf nodes (equal to the number of singleton indices) in the tree is $m_i$, where $1 \leq m_i \leq 2^{h_i}$. For example, in Fig. 6, the height of the tree is 3 and there are 5 leaf nodes. To compute the total number of bits broadcast by the reader in this round is equivalent to finding out the number $L_i$ of nodes in the polling tree (except the virtual root node), since each node corresponds to 1-bit information sent by the reader. By adding up the communication overhead in each round, we get the average length $w$ of the polling vector in TPP:

$$w = \frac{1}{n} \sum_{i=1}^{M} L_i, \tag{6}$$

where $M$ is the number of rounds needed to interrogate all tags. Clearly, the bigger the value of $\sum_{i=1}^{M} L_i$ is, the longer the polling vector is required for picking each tag. Our objective is thus to minimize $w$ to achieve high polling efficiency. However, directly deriving the minimal $w$ faces two challenges. First, even given $m_i$ and $h_i$ in the $i$th query round, $L_i$ cannot be determined as it varies with the tree structure that depends on uncertain distribution of singleton indices. Second, the number of unread tags in each inventory round relies on the previous round, i.e., $n_{i+1} = n_i - m_i$ $(1 \leq i \leq k)$, which makes it challenging to get an algebraic expression in closed form.

Therefore, we derive an upper bound of $w$ instead. Consider an arbitrary polling round, such as the $i$th round, $1 \leq i \leq k$. Let $w_i$ be the length of the polling vector in this round. Clearly, the maximum of $w_i$ is an upper bound of $w$. Given $m_i$, $w_i$ is proportional to $L_i$, as $w_i = \frac{L_i}{m_i}$. Let $L_i^+$ be the maximal $L_i$ when $m_i$ and $h_i$ are fixed. We can get $L_i^+$ when the tree tries to bifurcate as early as possible:

$$\begin{aligned} L_i^+ &= \sum_{i=1}^{k} 2^i + (h_i - k) \times m_i \\ &= 2^{k+1} - 2 + (h_i - k) \times m_i, \end{aligned} \tag{7}$$

Figure 7. Tree-based polling. (a) The reader broadcasts '000'; $A$ is polled. (b) The reader broadcasts '10'; $B$ is polled. (c) The reader broadcasts '1'; $C$ is polled. (d) The reader broadcasts '101'; $D$ is polled. (e) The reader broadcasts '11'; $E$ is polled.

where $k$ satisfies $2^k < m_i \leq 2^{k+1}$. Hence, we get the maximum $w_i^+$ of $w_i$:

$$w_i^+ = \frac{L_i^+}{m_i} = \frac{2^{k+1} - 2}{m_i} + (h_i - k), \tag{8}$$

where $k$ satisfies $2^k < m_i \leq 2^{k+1}$. Our goal is reduced to minimize $w_i^+$ by optimizing $h_i$. Assume that the probability that an index (singleton index) is picked by only one tag is $\mu$, $0 \leq \mu \leq 1$. We have $m_i = \mu \times 2^{h_i}$. Replacing $m_i$ in (8) with this expression, we have

$$w_i^+ = \frac{2^{k+1} - 2}{\mu \times 2^{h_i}} + (h_i - k), \tag{9}$$

where $k$ satisfies $2^k < \mu \times 2^{h_i} \leq 2^{k+1}$.

**Theorem 2.** *Given the index length $h_i$, the maximal length $w_i^+$ of the polling vector increases as $\mu$ decreases.*

*Proof:* Consider two probabilities $\mu_1$ and $\mu_2$, $\mu_1 < \mu_2$. We have

$$w_i^+(\mu_2) - w_i^+(\mu_1) = \left(\frac{2^{k_2+1} - 2}{\mu_2 \times 2^{h_i}} - \frac{2^{k_1+1} - 2}{\mu_1 \times 2^{h_i}}\right) + (k_1 - k_2), \tag{10}$$

where $k_1 \leq k_2$ and

$$\begin{cases} 2^{k_1} < \mu_1 \times 2^{h_i} \leq 2^{k_1+1} \\ 2^{k_2} < \mu_2 \times 2^{h_i} \leq 2^{k_2+1}. \end{cases}$$

According to (10), it is easy to prove that $w_i^+(\mu_1) > w_i^+(\mu_2)$. Hence, $w_i^+$ decreases as $\mu$ increases. ■

According to Theorem 2, we can get the minimal $w_i^+$ by maximizing the probability $\mu$. Given the number $n_i$ of unread tags, we have the number $m_i$ of singleton indices in this round according to (2):

$$m_i = n_i \times e^{-\frac{n_i - 1}{2^{h_i}}} \approx n_i \times e^{-\frac{n_i}{2^{h_i}}}. \tag{11}$$

Hence, the probability $\mu$ of a singleton index is

$$\mu = \frac{m_i}{2^{h_i}} = \frac{n_i}{2^{h_i}} \times e^{-\frac{n_i}{2^{h_i}}}. \tag{12}$$

Let $\lambda = \frac{n_i}{2^{h_i}}$; we have $\mu = \lambda e^{-\lambda}$. Deriving the first-order derivative of $\lambda e^{-\lambda}$ and letting it be 0, we obtain the maximal $\mu$ when $\lambda = n_i / 2^{h_i} = 1$. Fig. 8 depicts the probability $\mu$ with respect to $n_i / 2^{h_i}$. As we can see, $\mu$ peaks at the maximum $1/e$ when $n_i = 2^{h_i}$. In most cases, however, $n_i$ cannot be exactly equal to $2^{h_i}$. We thus need to find the optimal integer $h_i$ that maximizes $\mu$.

Since the probability $\mu$ monotonically increases when $\lambda \leq 1$, we can gradually adjust $h_i$ to make $\lambda = \lambda_1 \in (0.5, 1]$. The corresponding $\mu$ is denoted as $\mu(\lambda_1)$ that is greater than any $\mu(\lambda)$ where $\lambda \leq 0.5$. We then decrease $h_i$ by one and obtain $2\lambda_1$ together with $\mu(2\lambda_1)$. Obviously, $\mu(2\lambda_1)$ is greater than any $\mu(\lambda)$ where $\lambda > 2\lambda_1$. Therefore, max($\mu$)=max$\{\mu(\lambda_1), \mu(2\lambda_1)\}$. The max($\mu$) attains the minimum when $\mu(\lambda_1) = \mu(2\lambda_1)$. That is

$$\lambda_1 \times e^{\lambda_1} = 2\lambda_1 \times e^{2\lambda_1}. \tag{13}$$

According to (13), we have $\lambda_1 = \ln 2$. Therefore, given $n_i$, $\mu$ attains the maximum when $h_i$ satisfies:

$$\ln 2 \leq \frac{n_i}{2^{h_i}} < 2 \ln 2. \tag{14}$$

According to (14), we have the optimal index length $h_i$ meeting:

$$\log_2\left(\frac{n_i}{2 \ln 2}\right) < h_i \leq \log_2\left(\frac{n_i}{\ln 2}\right). \tag{15}$$

Hence, given the number $n_i$ of unread tags, we can derive the upper bound $w_i^+$ according to (8), (11), and (15). Although $w_i$ is determined by $n_i$, a rough upper bound of $w_i$ can be derived by minimizing max($\mu$), regardless of $n_i$. According to (13), $\min(\max(\mu)) = \ln 2 \times e^{-\ln 2} \approx 0.3466$, when $\lambda = \frac{n_i}{2^{h_i}} = \ln 2$. Substituting $0.3466$ for $\mu$ in (9), we have $k = h_i - 2$ and the upper bound of $w_i$:

$$w_i \leq w_i^+ \leq \frac{2^{h_i - 1} - 2}{0.3466 \times 2^{h_i}} + 2 < \frac{1}{0.3466 \times 2} + 2 \approx 3.44. \tag{16}$$

This is an upper bound of the length of the polling vector in an arbitrary inventory round, regardless of the number $n_i$ of unread tags. It therefore can be treated as the upper bound of the polling vector in TPP. Fig. 9 shows the average length $w$ of the polling vector with respect to the number $n$ of tags in TPP according to (6), (8), (11), and (15). The value of $n$ varies from 1000 to 100,000. We observe that $w$ remains stable at about 3.38 regardless of the number of tags, shortening the polling vector by a factor of 28 over CPP.

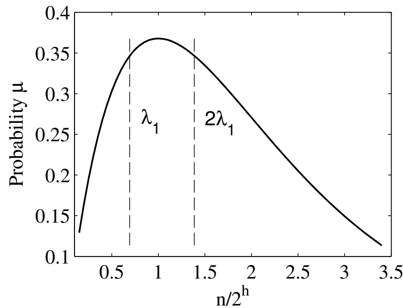


Figure 8. The probability $\mu$ with respect to $n/2^h$.

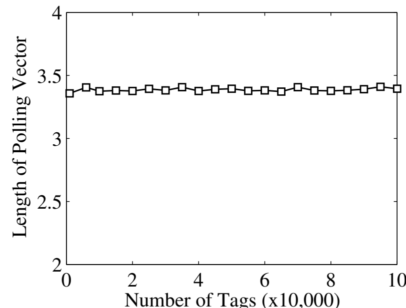


Figure 9. The average length $w$ of the polling vector with respect to the number $n$ of tags according to (6).

## V. SIMULATION

In this section, we evaluate the polling performance of HPP, EHPP, and TPP. We first compare the length of the polling vector of our protocols with that of CPP. We then apply these protocols to collect tag information and compare their execution time with the state-of-the-art information collection protocol.

### A. Simulation Setting

Our simulation settings follow the specification of the C1G2 standard [21]. Any two consecutive communications, from the reader to tags or vice versa, are separated by different time intervals. After the reader transmits any commands, all tags need to wait the transmit-to-receive turn-around time $T_1$ before replying to the reader. By contrast, after receiving the reply from tags, the reader has to wait the receive-to-transmit turn-around time $T_2$ before talking to tags. In the specification, $T_1$ is $\max(RTcal, 20T_{pri})$ and $T_2$ ranges from $3T_{pri}$ to $20T_{pri}$, where $RTcal$ is the reader-to-tag calibration symbol that equals the length of a data-0 symbol plus the length of a data-1 symbol, and $T_{pri}$ is the backscatter-link pulse-repetition interval. In our simulation, we set $T_1 = 100$ $\mu s$ and $T_2 = 50$ $\mu s$, which comply with the parameter configuration in the C1G2 standard.

The tag-to-reader transmission rate and the reader-to-tag data rate are not necessarily symmetric relying on the physical implementation and the practical environment. The transmission rate from tags to the reader depends on the data coding, 40 kbps to 640 kbps for FM0 and 5 kbps to 320 kbps for Miller-modulated subcarrier. We extract the intersection set 40 kbps to 320 kbps and adopt the lower bound 40 kbps as the data rate. In other words, it takes a tag 25 $\mu s$ to transmit one bit. The data rate from the reader to tags is from 26.7 kbps to 128 kbps. Similarly, we set the data rate to the lower bound 26.7 kbps, which takes the reader 37.45 $\mu s$ to transmit one bit. Note that other parameter settings may change the absolute metric, but the simulation conclusions can be drawn in a similar way.

Assume that $w$ is the length of the polling vector for sifting out a tag. The reader needs $(37.45{\times}(4 + w) + T_1 + 25{\times}\ell + T_2)$ $\mu s$ to collect $\ell$-bit information from a tag with our protocols, where $37.45{\times}(4{+}w)$ is the time for the reader to transmit a 4-bit $QueryRep$ command together with $w$-bit polling vector, $T_1$ is the waiting time before a tag replies, $25{\times}\ell$ is the period that a tag transmits $\ell$-bit information, and $T_2$ is the waiting time before the next polling. All results are the average outcome of 100 simulation runs in Java.

### B. Polling Vector

In Fig. 10, we compare the polling vector of HPP, EHPP, and TPP under various numbers of tags. For EHPP, the length of the circle command is set to 128 bits and the communication overhead to initiate each round of HPP is set to 32 bits. As shown in the figure, the polling vector of HPP almost sees a logarithmic growth over $n$. For instance, $w$ is closed to 9.5 when $n = 1000$, whereas $w$ is about 16 when $n = 10^5$. The main reason for this is that the bigger $n$ is, the longer the index is, leading to a longer polling vector. By contrast, EHPP remains stable at about 9.0 bits, without increasing with $n$. It not only shortens the polling vector but also keeps them steady compared with HPP. TPP further improves the polling efficiency, which levels off at only about 3.06 regardless of the number of tags. That is because TPP always can balance the ratio of the number of leaf nodes to the number of all nodes in the polling tree by dynamically adjusting the index length $h$. The bigger $h$ is, the more common prefixes tags share, ensuring that $w$ in each round is stable. Compared with the 96-bit polling vector in CPP, HPP takes about one fifth of the polling vector when $n{\leq}10^5$, EHPP and TPP respectively shorten the polling vector by a factor of 10 and 31, regardless of the number of tags.

### C. Execution Time Comparison

In this subsection, we apply HPP, EHPP, and TPP to collect tag information from all tags and compare their execution time with the state-of-the-art information collection protocol MIC [15]. In each simulation run, we collect three types of information: 1 bit, 16 bits, and 32 bits. Let $\ell$ be the length of tag information and $n$ be the number of tags. The lower bound of execution time for any

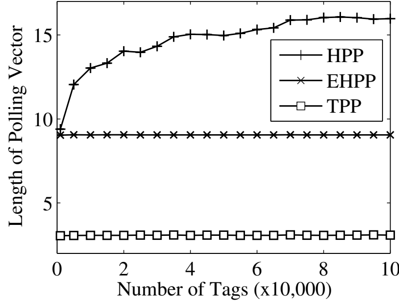


Figure 10. The average length $w$ of the polling vector with respect to the number $n$ of tags in HPP, EHPP, and TPP.

information collection protocol under the C1G2 standard is $(37.45\times4 + T_1 + 25\times\ell + T_2)\times n = (299.8 + 25\times\ell)\times n$ $\mu s$.

Table I compares the execution time of above five protocols when collecting 1-bit tag information which can be used to detect or identify missing tags against theft. The execution time of the protocols almost increases linearly with $n$ except for HPP. That is because HPP's polling vector increases with $n$, resulting in more polling overhead per tag for bigger $n$. We examine an arbitrary column in the table, such as the third column when n=10,000. CPP takes the longest time 37.70s that is about 10.64 times of the lower bound. HPP decreases the execution time to 8.12s as it avoids broadcasting 96-bit IDs. On top of HPP, EHPP reduces the execution time to 6.63s with multiple circles. Our best protocol TPP further improves the polling performance and its execution time drops to 4.39s which is only 1.35 times over the lower bound and decreases by 14.8% compared with 5.15s of MIC. Similar conclusions can be drawn in other columns: TPP performs the best, MIC follows, then EHPP and HPP, and finally CPP.

Table II and Table III show the execution time when tag information is 16 bits long and 32 bits long, respectively. The conclusion in Table I almost can be drawn. For example, to collect 16-bit information from 10,000 tags, the execution time of TPP is 85.7% of MIC, 78.3% of EHPP, 68.6% of HPP, and 19.6% of CPP. In Table III, when $n = 10,000$ and the information is 32 bits long, TPP needs 1.10 times of the lower bound, MIC requires 1.28 times of the lower bound, EHPP consumes 1.31 times of the lower bound, HPP is equal to 1.45 times of the lower bound, and CPP takes 4.14 times of the lower bound. Note that EHPP takes the same overhead as HPP does when the number of tags is 100, because EHPP in this small tag scale just executes HPP as-is. HPP works better than MIC when collecting 32-bit information from 100 tags, since the length of the polling vector is short when $n$ is small and the slot waste can also be totally avoided in HPP.

## VI. Related Work

Most existing protocols in RFID systems are ALOHA-based [12], [11], [15], [16], [24]. Tan et al. [11] design

Table I
EXECUTION TIME COMPARISON (IN SECONDS) TO COLLECT 1-BIT INFORMATION

| | n=$10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|
| CPP | 0.377 | 3.770 | 37.70 | 377.02 | 3770.2 |
| HPP | 0.054 | 0.677 | 8.12 | 92.29 | 1051.5 |
| EHPP | 0.054 | 0.664 | 6.63 | 66.39 | 663.8 |
| MIC, k=7 | 0.049 | 0.514 | 5.15 | 51.48 | 515.3 |
| **TPP** | 0.043 | 0.440 | 4.39 | 44.03 | 439.8 |
| LowerBound | 0.032 | 0.324 | 3.24 | 32.48 | 324.8 |

Table II
EXECUTION TIME COMPARISON (IN SECONDS) TO COLLECT 16-BIT INFORMATION

| | n=$10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|
| CPP | 0.414 | 4.145 | 41.45 | 414.52 | 4145.2 |
| HPP | 0.092 | 1.053 | 11.87 | 129.79 | 1426.7 |
| EHPP | 0.092 | 1.039 | 10.39 | 103.92 | 1038.8 |
| MIC, k=7 | 0.090 | 0.953 | 9.50 | 95.25 | 950.9 |
| **TPP** | 0.080 | 0.815 | 8.14 | 81.55 | 814.6 |
| LowerBound | 0.069 | 0.699 | 6.99 | 69.98 | 699.8 |

Table III
EXECUTION TIME COMPARISON (IN SECONDS) TO COLLECT 32-BIT INFORMATION

| | n=$10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|
| CPP | 0.454 | 4.545 | 45.45 | 454.52 | 4545.2 |
| HPP | 0.132 | 1.451 | 15.89 | 169.79 | 1826.5 |
| EHPP | 0.132 | 1.440 | 14.39 | 143.89 | 1438.9 |
| MIC, k=7 | 0.141 | 1.398 | 14.15 | 141.53 | 1415.4 |
| **TPP** | 0.121 | 1.213 | 12.14 | 121.52 | 1214.8 |
| LowerBound | 0.109 | 1.099 | 10.99 | 109.98 | 1099.8 |

the Trust Reader Protocol (TRP) and the Untrusted Reader Protocol (UTRP) to detect the missing-tag event with the probability $\alpha$ by checking the 0-1 status of tags' reply slots, without collecting tag IDs. Li et al. [12] propose a series of efficient protocols that not only detect the missing-tag event with certainty but also tell exactly which tags are missing. Although they achieve high performance by eliminating potential collisions in advance, the useless empty slots cannot be avoided in their protocol design. To alleviate this waste, Chen et al. [15] propose an efficient Multi-hash Information Collection Protocol (MIC) by mapping multiple hash functions. Compared with basic ALOHA-based protocols, MIC decreases the wasted slots (empty slots and collision slots) from 63.2% to 13.9% in a query round when 7 hash functions are used, greatly improving the performance of information collection. Increasing the number of hash functions will further reduce the slot waste. However, this is not free; the large number of hashes increases the size of the indicator vector sent by the reader and puts more storage burden on the tags. This dilemma is essentially due to the choice of ALOHA-based approaches.

Polling, as a classic anti-collision protocol, provides an intuitive request-response way to interrogate RFID tags. Because the reader's request and the tag's reply are a

one-to-one mapping, polling can totally remove slot waste. However, broadcasting 96-bit tag IDs for isolating each tag causes high communication overhead. To improve the query efficiency, Li et al. [12] and Qiao et al. [19] both propose advanced polling-assisted protocols for missing tag identification and tag information collection, respectively. By polling a part of tags in collision slots, they can convert the useless collision slots into useful singleton slots, thereby saving the communication overhead. However, these protocols target at reducing the number of polling; the polling vector during each polling still adopts tedious tag IDs. The most related work to improve the polling efficiency is the Coded Polling protocol (CP) that reduces the length of the polling vector by half through validating cyclic redundancy code [19]. Even so, CP is far from a time-efficient polling protocol as the 48-bit polling vector is still too long for picking a tag.

## VII. Conclusion

Polling, as a widely-used anti-collision protocol, plays an important role in interrogating RFID tags. Conventional polling, however, needs to separate each tag from others by transmitting tedious tag IDs, which is time-consuming in large RFID systems. In this paper, we make fundamental improvements on the polling protocol design by shortening the polling vector. Both theoretical analysis and simulation results show that our best protocol can dramatically decrease the polling vector from 96 bits to only about 3 bits, regardless of the number of tags. Besides, we apply our protocols to collect tag information. The simulation results demonstrate our best protocol is faster than the state-of-the-art protocol under the specification of the C1G2 standard.

## Acknowledgment

This research is financially supported by the National Natural Science Foundation of China (No.61272418), the National Science and Technology Support Program of China (No.2012BAK26B02), the Future Network Prospective Research Program of Jiangsu Province (No.BY2013095-5-02), the Lianyungang City Science and Technology Project (No.CG1420,JC1508), the Fundamental Research Funds for the Central Universities, and HK PolyU B-Q38F.

## References

[1] L. Ni, Y. Liu, Y. C. Lau, and A. Patil, "LANDMARC: Indoor location sensing using active RFID," in *Proc. of IEEE PerCom*, 2003, pp. 407–415.

[2] L. Yang, Y. Chen, X.-Y. Li, C. Xiao, M. Li, and Y. Liu, "Tagoram: Real-time tracking of mobile RFID tags to high precision using cots devices," in *Proc. of ACM Mobicom*, 2014, pp. 237–248.

[3] C. Wang, L. Xie, W. Wang, and S. Lu, "Moving tag detection via physical layer analysis for large-scale RFID systems," in *Proc. of IEEE INFOCOM*, 2016.

[4] L. Xie, Q. Li, X. Chen, S. Lu, and D. Chen, "Continuous scanning with mobile reader in RFID systems: An experimental study," in *Proc. of ACM MobiHoc*, 2013, pp. 11–20.

[5] X. Liu, S. Zhang, B. Xiao, and K. Bu, "Flexible and time-efficient tag scanning with handheld readers," *IEEE Transactions on Mobile Computing*, vol. 15, no. 4, pp. 840–852, 2016.

[6] J. Liu, B. Xiao, K. Bu, and L. Chen, "Efficient distributed query processing in large RFID-enabled supply chains," in *Proc. of IEEE INFOCOM*, 2014, pp. 163–171.

[7] Y. Yin, L. Xie, J. Wu, and S. Lu, "Focus and shoot: Exploring auto-focus in rfid tag identification towards a specified area," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 888–901, 2016.

[8] X. Liu, B. Xiao, S. Zhang, and K. Bu, "Unknown tag identification in large RFID systems: An efficient and complete solution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1775–1788, 2015.

[9] J. Liu, B. Xiao, S. Chen, F. Zhu, and L. Chen, "Fast RFID grouping protocols," in *Proc. of IEEE INFOCOM*, 2015, pp. 1948–1956.

[10] X. Liu, K. Li, G. Min, K. Lin, B. Xiao, Y. Shen, and W. Qu, "Efficient unknown tag identification protocols in large-scale RFID systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3145–3155, 2014.

[11] C. C. Tan, B. Sheng, and Q. Li, "How to monitor for missing RFID tags," in *Proc. of IEEE ICDCS*, 2008, pp. 295–302.

[12] T. Li, S. Chen, and Y. Ling, "Identifying the missing tags in a large RFID system," in *Proc. of ACM MobiHoc*, 2010, pp. 1–10.

[13] K. Bu, J. Liu, B. Xiao, X. Liu, and S. Zhang, "Intactness verification in anonymous RFID systems," in *Proc. of IEEE ICPADS*, 2014, pp. 134–141.

[14] X. Liu, K. Li, G. Min, Y. Shen, A. X. Liu, and W. Qu, "Completely pinpointing the missing RFID tags in a time-efficient way," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 87–96, 2015.

[15] S. Chen, M. Zhang, and B. Xiao, "Efficient information collection protocols for sensor-augmented RFID networks," in *Proc. of IEEE INFOCOM*, 2011, pp. 3101–3109.

[16] H. Yue, C. Zhang, M. Pan, Y. Fang, and S. Chen, "A time-efficient information collection protocol for large-scale RFID systems," in *Proc. of IEEE INFOCOM*, 2012, pp. 2158–2166.

[17] H. Vogt, "Efficient object identification with passive RFID tags," in *Proc. of IEEE PerCom*, 2002, pp. 98–113.

[18] X. Liu, B. Xiao, K. Li, J. Wu, A. X. Liu, H. Qi, and X. Xie, "RFID cardinality estimation with blocker tags," in *Proc. of IEEE INFOCOM*, 2015, pp. 1679–1687.

[19] Y. Qiao, S. Chen, T. Li, and S. Chen, "Energy-efficient polling protocols in RFID systems," in *Proc. of ACM MobiHoc*, 2011, pp. 25:1–25:9.

[20] Y. Zheng and M. Li, "P-MTI: Physical-layer missing tag identification via compressive sensing," in *Proc. of IEEE INFOCOM*, 2013, pp. 917–925.

[21] *Epcglobal. epc radio-frequency identity protocols class-1 generation-2 uhf rfid protocol for communications at 860 mhz-960mhz version 1.2.0*, Tech. Rep., 2008.

[22] M. Buettner and D. Wetherall, "An empirical study of UHF RFID performance," in *Proc. of ACM MobiCom*, 2008, pp. 223–234.

[23] T. Li, S. Wu, S. Chen, and M. Yang, "Energy efficient algorithms for the RFID estimation problem," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.

[24] S.-R. Lee, S.-D. Joo, and C.-W. Lee, "An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification," in *Proc. of MobiQuitous*, 2005, pp. 166–172.