# On Improving Write Throughput in Commodity RFID Systems

Jia Liu† Xingyu Chen† Xiulong Liu‡ Xiaocong Zhang† Xia Wang† Lijun Chen†
†State Key Laboratory for Novel Software Technology, Nanjing University, China
‡Department of Computing, The Hong Kong Polytechnic University, China
Email: jialiu@nju.edu.cn, {xychen, wangxia}@smail.nju.edu.cn, xiulongliudut@gmail.com, chenlj@nju.edu.cn

***Abstract*—High write throughput plays a vital role in improving the time efficiency of RFID-enabled applications, such as password update and over-the-air programming. In this paper, we make a fresh attempt to study the under-investigated problem of *group write* and discuss how to improve the write throughput in the commodity RFID system. By conducting extensive experiments, we reveal that the select operation specified by the C1G2 standard takes up the large overhead of a write cycle, which is the key factor in determining the write efficiency. Based on this finding, we propose an efficient write bundling (WB) scheme that bundles multiple writes up and executes them together in a burst mode, which greatly reduces the number of selects and thereby amplifies the write throughput. In WB, a carefully designed bit vector is assigned to each tag for connecting multiple tags into a run, such that a single select command is able to pick these tags concurrently. Besides, WB integrates multiple select commands into one by running a series of logic operations on the inventoried flags, which are used to indicate whether or not a tag is active. We implement WB in a commodity RFID system, with no need of any software modifications or hardware argument. Extensive experiment results show that WB is able to reduce the number of selects by 87.5% and produce a 2× write amplification, compared with the C1G2-compatible exclusive write.**

## I. Introduction

Radio frequency identification (RFID) is becoming ubiquitously available in our daily lives, including object tracking [1]–[5], library inventory [6], warehouse control [7]–[13], supply chain management [14] etc. This paper studies an under-investigated problem, improving the throughput of *group write* in commodity RFID systems, which is of practical importance for a variety of applications. Group write, as the name implies, is to write a piece of information into a group of RFID tags of interest, which is quite common in practice. For example, consider a supermarket where a batch of goods come in. The administrator needs to write specific information (e.g., the manufacturer, expiry date) into the tag's memory for the purpose of live query by the customers. In another example, the use of access password in RFID systems allows us to prevent unauthorized readers from accessing the tags' information in privacy-sensitive environment. However, the access password specified by the C1G2 RFID standard [15] is too weak to guard against the brute force attack. We may need to frequently update the passwords on tags to strengthen the privacy protection. Assigning each tag a different password is time-consuming. Alternatively, a common way is to split the entire tag set into groups, and assign the tags in the same group the same new password. In this scenario, the group write is required for improving the update efficiency of passwords. Finally, consider the function of over-the-air programming in RFID systems. Tags with the same purposes will be programmed in the same way, namely, the data or codes written to the tag memory of a group of tags are the same. With the ability of group write, the air programming will get a great performance gain.

In spite of growing importance, little work investigates how to improve the write throughput. The most related research is tag grouping [16]–[18], which aims to inform all tags about which groups they belong to, such that tags in the same subset will have the same group ID. With this group ID, the reader can simultaneously transmit the data to a group of tags, making the multicast possible in theory. In practice, however, tag grouping is out of the scope of the C1G2 RFID standard [15] and cannot be supported by commodity RFID systems. The essential reason is that one-to-many data transmission (which is the basis of the existing grouping protocols) is incompatible with the RFID standard.

In light of this, this paper studies the under-investigated problem of group write and aims to improve the write throughput in commodity RFID systems. According to the C1G2 standard, an intuitive solution to the group-write problem is exclusively writing (EW) each tag; one tag at a time. This works but suffers from high latency. That is because a write cycle comprises a trilogy: isolating a single tag from others (select operation), identifying the tag's ID (inventory operation), and finally writing the data into the tag's memory (write operation). It is time-consuming to run the write cycle repetitively for a group of tags to be written. To reduce this overhead, we conduct extensive experiments and in surprise, we find that the major overhead of a write cycle is caused by the select operation, which takes up more than half (about 60%) of the execution time of a write cycle. In addition, according to the real experiments and the specifications of C1G2, the inventory and write operations are mandatory in a write cycle, while the select operation is optional. In other words, it is possible to remove the select from a write cycle.

The above two findings shed light on the basic idea of our protocol design: reducing the number of select operations. This is, however, not easy. The select operation serves as a filter that picks the wanted tags. If we remove the selects directly, we may fail to deal with the tag subset of interest. Therefore, we propose an efficient write bundling (WB) scheme that bundles

the multiple writes up and executes them together in a burst mode, which greatly reduces the number of select operations and thereby improves the write efficiency. WB consists two phases: *tag bundling* and *select combination*. The former assigns each tag a carefully designed bit vector that is used for bundling multiple tags into a run and later picking these tags via one select operation. Select combination integrates multiple select operations into one by running a series of logic operations on the inventoried flags, which determine whether or not a tag participates in the incoming write cycle. With above two phases, WB is able to greatly reduce the number of selects, with the guarantee that all tags of interest are included while others are excluded. For example, when the number of tags to be written is the half of all tags, WB can decrease the number of selects by 87.5%. Besides, we implement WB in the commodity RFID system. Extensive experiments show that WB is far superior to the C1G2-compatible exclusive write. The main contributions of this work are three-fold.

• We formulate the under-investigated problem of group write and propose an efficient solution to this problem. As far as we know, this is the first work that studies how to improve the write throughput in commodity RFID systems, which is vital to a variety of applications, such as password update and over-the-air programming.

• We reveal the fact that the select operation takes up the major overhead of a write cycle and the reduction in the number of selects is the key to amplify the write throughput. Following this principle, we design an efficient write bundling (WB) scheme that improves the write performance through the technologies of tag bundling and select combination.

• We implement WB in a commodity RFID system, with no need of any software modifications or hardware augment. Extensive experiments show that WB is able to produce a $2\times$ write amplification, compared with the exclusive write.

## II. Problem Formulation

### A. Problem Definition

We consider an RFID system that consists of a reader and a number of tags. The reader is connected with a backend server for information storage and computation. Each tag has a unique tag ID that exclusively indicates the associated object. These tags can communicate with the reader directly, but not amongst themselves. In practice, one may need to write some specific information (manufacturers, product name, expiry date etc.) into a group of tags of interest. For example, the administrator of a supermarket is required to write the expiration date into tagged milk for the purpose of live query by the customers. Since the C1G2 RFID standard [15] specifies that tag's memory update is a one-to-one operation, writing tags individually brings high latency, especially in a large RFID system.

In this paper, we investigate the problem of how to write a piece of information into a group of tags, which is referred to as the *group write* problem. Specifically, given a tag set $\Gamma = \{t_1, t_2, ..., t_n\}$ and a subset $\Gamma' = \{t'_1, t'_2, ..., t'_m\}$ of $\Gamma$, the group write is to load the same data into each tag of $\Gamma'$. The tag subset $\Gamma'$ is determined by the user demands or upper-layer application requirements. For a generalized case, we do not make any assumption on the tag distribution of $\Gamma'$. The objective is to maximize the write throughput in a commodity RFID system, which helps minimize the application latency and improve the quality of service. Note that, if there are different data to be written, we can split the tag set into different groups first according to the content of data (the tags to be written the same data belong to the same group), and later execute the group write on each group.

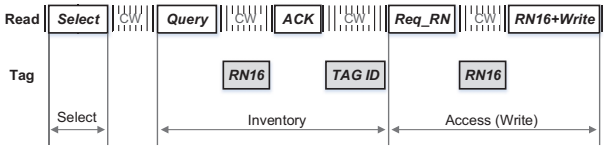



Fig. 1: Three operations of a write cycle in C1G2 [15].

### B. Existing Solution: Exclusive Write

The Class1 Gen2 (C1G2) protocol [15] is a worldwide UHF RFID standard that defines the physical interactions and logical operating procedures between the commodity readers and tags. Following C1G2, the RFID reader manages tag populations with three basic operations: *Select*, *Inventory*, and *Access*, which run in sequence. The select operation is prior to inventory. It allows a reader to choose a specific subset of tags that participate in the subsequent query round. After that, inventory carries out a slotted frame for tag identification (i.e., collecting tag IDs). Followed by inventory, access is the process by which a reader transacts with (including *write* or other engages with) an individual tag. Fig. 1 illustrates the three operations involved in a write cycle.

Therefore, to achieve the task of group write within the scope of C1G2, we can execute the write operation on each tag of $\Gamma'$ exclusively (*exclusive write, EW*) via a trilogy: isolating each tag from others (select operation), identifying its ID (inventory operation), and finally getting access to the single tag's memory (write operation). Since the C1G2 standard allows only one tag to be written at a time, there are totally $m$ writes needed for the group write of $\Gamma' = \{t'_1, t'_2, ..., t'_m\}$. Next, we reveal the weakness of the EW-based solution and propose a write bundling (WB) scheme to improve the write throughput in the commodity RFID system.

## III. Write Bundling Scheme

### A. Basic Idea

As aforementioned, a cycle of memory write on a tag comprises three operations: select, inventory, and write. Now we set up a scale-down RFID system to measure the time overhead of each operation. In the experiment, a reader and 5 tags are deployed. For each tag, we write a piece of information (16 bits long) into each tag's memory and repeat it 100 times. Fig. 2 shows the time periods of the three operations on each tag. As we can see, the inventory and the write take 8.9ms and 9.1ms, respectively. The select spends the longest time, with the mean of 28.4ms.

With above experiment results, we can see that the select takes up more than 60% of the execution time of a write cycle, while the other two operations (inventory and write) consume the left. If any of these overhead (especially for select's overhead) can be reduced, the overall write throughout will get a big boost. According to C1G2, the inventory and write operations are mandatory: any conforming readers and tags shall do the inventory-write operation when updating the tag's memory. Therefore, the overhead caused by inventory and write cannot be avoided. On the contrary, the select operation is optional, which provides us with the chance to reduce the number of selects and thereby improve the write efficiency. This forms the basic idea of our protocol design.

We conduct a feasibility study that evaluates the potential of this idea. In the experiment, an Alien reader (ALR F800) and a number of Alien Higgs4-based tags (ALN 9740) [19] are deployed. These Alien devices support a capability of `BlastWrite`™ , which allows the reader to perform the inventory and write operations on a set of tags, with no need of selects. This helps the reader avoid the broadcast of selects (that takes up more than half of the execution time of a memory write). In Fig. 3, we compare EW-based solution with the blastwrite. We write 32-bit data into all tags in the field of view (i.e., $\Gamma' = \Gamma$). As we can see, both of the two solutions experience a linear increase with the number of tags. The difference is that blastwrite is more efficient, which produces a 2.2× write amplification in the ballpark.

In above study, we just consider the group write in the specific case of $\Gamma' = \Gamma$, where all on-site tags are taken into account and selects are not needed. However, in a more generalized case of $\Gamma' \subset \Gamma$, we need to separate $\Gamma'$ from $\Gamma$, such that the group write is conducted on the tag subpopulation of interest, instead of all. This is not easy in practice. If nothing is done, the method degrades to the EW-based solution, in which each tag corresponds to a select and $m$ selects are required in total. Next, we present our protocol: *write bundling (WB)*, which reduces the number of selects by bundling tags up and combining multiple selects together.

### B. Design of WB

The basic idea of WB is that each time the reader picks multiple tags from the interested tag set $\Gamma'$ via only one select operation, and then executes the blastwrite on the selected tags. Since the selected tags are dealt in a burst mode, the number of select operations can be sharply decreased. Next, we first describe how a select operation works within the scope of the C1G2 standard, and then discuss the details of WB.

*1) Select Function:* C1G2 defines a mandatory command called `Select` that allows a reader to choose a specific tag subset that participates in the incoming inventory-write operation. It consists of six fields.

• `MemBank, Pointer, Length, Mask.` These four fields jointly determine which tags are matching or not. `MemBank` specifies the memory bank for comparison. There are four distinct memory banks (page 41 in [15]).

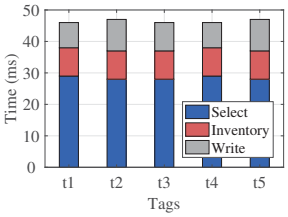


Fig. 2: Time of 3 operations.

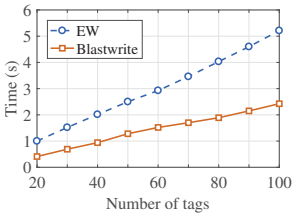


Fig. 3: EW vs. blastwrite.

`MemBank-0` is reserved for kill and access passwords if encryption is implemented on the conforming tag. `MemBank-1` stores the Electronic Product Code (EPC), i.e., tag ID that is often referred to. `MemBank-2` stores `TID` that indicates the tag- and manufacturer-specific data at the time of manufacture, which is permalocked and unchangeable. `MemBank-3` is user memory that allows customized data write and access. `Pointer` indicates the starting position in the chosen memory bank. `Length` determines the length of `Mask`, which is a customized bit string according to the user demands. If `Mask` is the same as the string that begins at `Pointer` and ends `Length` bits later in the memory of `MemBank`, the corresponding tag is matched.

• `Target, Action.` The field `Target` indicates the object that `Select` will operate. It is either a tag's selected flag (SL) or an inventoried flag, which is a one-bit indicator that serves as the access control of a tag. In other words, the selection function is actually achieved by masking the interested tags, setting the matching tags' flags to a specific value while not-matching tags to opposite, and finally operating the tags with the same flag value. How to set the flag is determined by `Action` field. As show in Table I, there are eight actions, where matching and not-matching tags assert or dessert their SL flags, or set their inventoried flags to $A$ or $B$. By combining `Target` and `Action`, the reader is able to modify the specific flag of a group of tags.

Fig. 4 illustrates the `Select` command. There are total 6 tags in the field of view. The reader issues a `Select` command to let the tags $t_2$, $t_4$, $t_6$ (highlighted with dark gray) whose data starting at the third bit with a length of 2 bits in the MemBank-1 equal $01_2$, set their inventoried flags to $A$, while other tags (with light gray color) that do not match the mask set their inventoried flags to $B$. With these settings, the reader is able to deal with the tags with the inventoried flag equal to $A$, namely, these tags with $A$ are selected. C1G2 specifies that a tag shall support four inventoried flags, each of which corresponds to a session, which is used to fit the case of exclusive reading amongst multiple readers. In the

TABLE I: Eight Actions of `Select`.

| Action | Tag Matching | Tag Not-Matching | Abbr. |
|---|---|---|---|
| 000 | assert **SL** or **inventoried** → *A* | deassert **SL** or **inventoried** → *B* | AB |
| 001 | assert **SL** or **inventoried** → *A* | do nothing | A- |
| 010 | do nothing | deassert **SL** or **inventoried** → *B* | -B |
| 011 | negate **SL** or (*A*→*B*, *B*→*A*) | do nothing | S- |
| 100 | deassert **SL** or **inventoried** → *B* | assert **SL** or **inventoried** → *A* | BA |
| 101 | deassert **SL** or **inventoried** → *B* | do nothing | B- |
| 110 | do nothing | assert **SL** or **inventoried** → *A* | -A |
| 111 | do nothing | negate **SL** or (*A*→*B*, *B*→*A*) | -S |

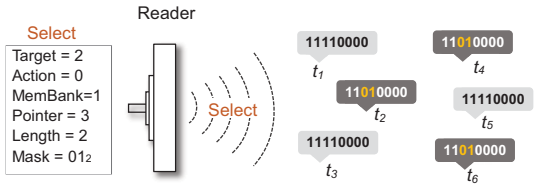



Fig. 4: An illustration of Select.

example of Fig. 4, the inventoried flag in session 2 is adopted as `Target` is set to $010_2$ (see [15]). The other inventoried flags can also be used in the similar way. Besides, if SL is used (`Target`=$100_2$), we can implement the select function by asserting SL of the tags of interest and deasserting that of others. For ease of presentation, we take the inventoried flag as the metric to show our protocol design in what follows.

*2) Reduction in Selects:* Although `Select` provides us with the special ability of picking tags, selecting a group of tags with a single select command is not easy. Take the Fig. 4 for example. Assume that the interested tag set $\Gamma'$ is $\{t_2, t_3, t_4\}$. Since they do not comprise any common substring at the same position of the memory bank, we cannot find any common mask to uniformly select them via one transmission. Instead, we need to deal with each of them separately; three selects are needed. This is time consuming and is very likely to happen in practice, especially in a large RFID system where a common mask can hardly meet the condition that wanted tags are included while others are excluded. Hence, two approaches are proposed to reduce the number of selects: tag bundling and select combination.

*a) Tag Bundling:* The reason why selecting multiple tags in one transmission is challenging is that any tag distribution of $\Gamma'$ is possible and there is no guarantee that a single mask is able to uniformly isolate these tags from others. Instead of directly using tag IDs, we assign each tag a carefully designed bit vector to increase the probability of one-to-more select (one command selects multiple tags), which reduces overall communication overhead. As aforementioned, a tag's memory comprises four banks, in which MemBank-3 allows the user to write. By pre-loading a bit vector in this memory space, we can use it to reduce the number of selects. The key is how to design the bit vector. Consider the entire tag set $\Gamma = \{t_1, t_2, ..., t_n\}$. The bit vector is $2n$ bits long, each of which is initialized to 0 first, where $n$ is the number of tags. For the $i$-th tag $t_i$, without loss of generality, we set both of the $i$-th bit and the $(n+i)$-th bit of the vector to '1'; the left stay the same (i.e., '0'), which is shown in Fig. 5.

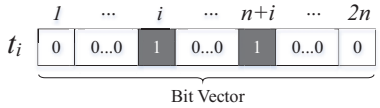



Fig. 5: The bit vector assigned to the $i$-th tag $t_i$.

By this means, any consecutive tags can be chosen by a select. Consider a tag set $\Gamma_{ij} = \{t_i, t_{i+1}, ..., t_j\}$ with consecutive tags starting from $t_i$ to $t_j$, $1 \le i \le j \le n$. To select these $(j-i+1)$ tags, we just need to check the consecutive bits of the vector, from the $(j+1)$-th bit to the $(n+i-1)$-th bit. If each bit of the substring is equal to '0', a tag is selected. Otherwise, the tag is excluded. The reason why this works is that none of the bits from $(j+1)$ to $(n+i-1)$ is equal to '1' if a tag belongs to $\Gamma_{ij}$. On the contrary, there must be exactly one bit set to '1' by the reader if a tag is out of $\Gamma_{ij}$. In this way, the tag set $\Gamma_{ij}$ (consecutive tags from $t_i$ to $t_j$) are included whereas all other tags are excluded. Unlike exclusive select, only one select operation rather than $(j-i+1)$ operations is needed, which saves the communication overhead. As shown in Fig. 6, there are six tags $\Gamma = \{t_{1-6}\}$ and the bit vector is 12 bits long. For the $i$-th tag $t_i$, the $i$-th bit and $(6+i)$-th bit are set to '1'. If $\Gamma_{ij} = \Gamma_{34} = \{t_3, t_4\}$ is needed to be selected, we just check the consecutive bits from the position of $4+1=5$ to that of $6+3-1=8$. More specifically, only $\Gamma_{34} = \{t_3, t_4\}$ owns the common substring $0000_2$ that starts at the 5th bit and ends at the 8th bit. We can use this substring to select them and others will be excluded.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t_1$ | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $t_2$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| $t_3$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $t_4$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $t_5$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $t_6$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Fig. 6: Bit vectors and the selection of $\Gamma_{34} = \{t_3, t_4\}$.

To carry out this selection in C1G2, we need to build a specific `Select` command, which is denoted by:

$$\mathcal{S}(\underbrace{t}_{\texttt{Target}}, \overbrace{a}^{\texttt{Action}}, \underbrace{b}_{\texttt{MemBank}}, \overbrace{p}^{\texttt{Pointer}}, \underbrace{l}_{\texttt{Length}}, \overbrace{k}^{\texttt{Mask}}), \quad (1)$$

where $t$, $a$, $b$, $p$, $l$, $k$ indicate the fields of `Target`, `Action`, `MemBank`, `Pointer`, `Length`, and `Mask`, respectively. Picking the consecutive tags from $t_i$ to $t_j$ is equivalent to setting these tags' flags to the same value ($A$ for inventoried flag, *assert* for SL) while others' flags to opposite ($B$ for inventoried flag, *deassert* for SL). The `Select` command for picking $\Gamma_{ij}$ is:

$$\Gamma_{ij} : \mathcal{S}(2,\ 0,\ 3,\ j+1,\ n+i-j-1,\ 0),$$

where $t = 2$ sets the target to the inventoried flag of session 2, $a = 0$ sets the flags of matching tags to $A$ while not-matching tags to $B$ (see Table I), $b = 3$ determines the MemBank-3, $p = j+1$ and $l = n+i-j-1$ indicate the memory position for mask matching, and the mask is $l$ '0's ($k = 0$). After this selection, all tags in $\Gamma_{ij}$ are set to $A$ while others out of $\Gamma_{ij}$ are set to $B$. We can execute the blastwrite on the selected $\Gamma_{ij}$ to achieve the group write task. Take Fig. 6 for example. The select command for picking $\Gamma_{34}$ is $\mathcal{S}(2,\ 0,\ 3,\ 5,\ 4,\ 0000_2)$.

The use of bit vector helps us select consecutive tags with only one transmission. For any interested tag set $\Gamma' = \{t'_1, t'_2, ..., t'_m\}$, however, there is no guarantee that all tags in $\Gamma'$ are consecutive. Hence, multiple selects might be required. We refer to a tag subset in which the tags' indices are consecutive (the tags out of the subset are not next to any tag in the subset) as a *run*. For example, if the tag set $\Gamma'$

of interest is $\{t_2, t_3, t_4, t_6, t_7\}$, there are two runs: $\{t_2, t_3, t_4\}$ and $\{t_6, t_7\}$. Since all tags in a run are consecutive, a select is sufficient to pick a run. The number of runs is actually the number of selects to be carried out. Since a run comprises one or multiple tags, the number of selects is less than that of tags, which reduces the selection overhead, compared with the exclusive select. For example, the above example reduces the number of selects from 5 to 2, getting a 60% decrease.

*b) Select Combination:* Tag bundling clusters multiple tags to form a run, which makes one-to-more select available, reducing the communication overhead. Unlike tag bundling, select combination targets at combining multiple selects together in one transmission to improve the write performance. Specifically, by investigating commodity RFID readers through their data sheets and real experiments, we find that these readers allow a select command to contain multiple masks, e.g., up to two masks are supported by Impinj R420 [20] and four masks by ALR 9900+ and ALR F800 [19]. With this function, we are able to fill several masks into one select, such that multiple original selects are compressed into a single one, greatly saving the communication overhead. Now, we detail how to configure multiple masks in one `Select` to make them tailored to a specific tag set.

Assume that there are $r$ runs in $\Gamma'$ after tag bundling. If no extra action is taken, the reader needs to carry out $r$ select commands. Instead, we bundle multiple selects together. Take Alien readers (ALR 9900+ and ALR F800 [19]) that offer up to four masks in a select for example. We need two kinds of select actions: $AB$ and $A-$ (see Table I). The former means the matching tags set their inventoried flags to $A$ whereas the not-matching tags move their inventoried flags to $B$. Similarly, the action $A-$ also sets the matching tags to $A$. The difference is that not-matching tags do nothing to their inventoried flags. Consider any four runs $\Gamma'_1, \Gamma'_2, \Gamma'_3, \Gamma'_4$ that need to be selected. We set the masks as follows:

$$\begin{aligned} &① \ \Gamma'_1 \leftarrow AB: \ \mathcal{S}(2,0,3,p_1,l_1,0),\\ &② \ \Gamma'_2 \leftarrow A-: \ \mathcal{S}(2,1,3,p_2,l_2,0),\\ &③ \ \Gamma'_3 \leftarrow A-: \ \mathcal{S}(2,1,3,p_3,l_3,0),\\ &④ \ \Gamma'_4 \leftarrow A-: \ \mathcal{S}(2,1,3,p_4,l_4,0), \end{aligned} \tag{2}$$

where $p_i$ and $l_i$ indicate the matching position of the run $\Gamma'_i$. As shown in (2), the first step is to set the inventoried flags of $\Gamma'_1$ to $A$, while move those of other tags to $B$ (`Action`=$AB$ is adopted). In the second step, we need to update $\Gamma'_2$ to $A$ but not change $\Gamma'_1$. Hence, the action of $A-$ is taken such that only the matching $\Gamma'_2$ is set to $A$ while other not-matching tags stay the same. Similarly, the step ③ and ④ move $\Gamma'_3$ and $\Gamma'_4$ to $A$ respectively; no changes happen on other not-matching tags. Note that these four steps just specify the masks and actions to be taken in a select; the reader has not sent the select command out yet. The select is actually issued when an inventory round begins. In this way, by loading four masks into one select command, the number of selects is dropped by 75%, greatly saving the communication overhead. One may concern whether or not broadcasting the select with multiple masks takes longer time than that with only a single mask. The answer is yes but the extra overhead is negligible. That is because, besides masks, a select command comprises many other overhead, such as the preamble, frame-sync, CRC, communication interval [15], which make the masks take up only a small portion of the transmission overhead of a select. This will be discussed shortly later in Section V-C.

### C. Performance Analysis

Next, we discuss how many selects can be reduced by WB with the help of tag bundling and select combination. As aforementioned, tag bundling makes the number of selects equal to the number of runs. We now analyze the expected value of the number of runs. Considering the entire tag set $\Gamma$ with $n$ tags, a specific $\Gamma'$ can be presented by an $n$-bit vector, where the $i$-th bit corresponds to the tag $t_i$: '1' indicates $t_i \in \Gamma'$ and '0' indicates $t_i \notin \Gamma'$. For example, if $n = 10$ and $\Gamma' = \{t_2, t_3, t_7, t_8, t_9\}$, the equivalent vector is $0110001110_2$, which has two runs of '1's. The problem is reduced to derive the number of runs of '1's in the vector.

Suppose that each tag of $\Gamma'$ is randomly picked from $\Gamma$. There are $\binom{n}{m}$ outcomes of $n$ tags taken $m$ in total. Given a number $r$, $1 \le r \le n$, we now determine the number of vectors that exactly have $r$ runs of '1's. Let each run have $x_i$ '1's, $1 \le i \le r$. We have $x_1 + x_2 + \cdots + x_r = m$. If we let $y_1$ denote the number of '0's before the first run of '1's, $y_2$ the number of '0's between the first 2 runs of '1's, ... , $y_{r+1}$ the number of '0's after the last run of '1's, then $y_i$ satisfies:

$$\sum_{i=1}^{r+1} y_i = n - m, \tag{3}$$

where $y_1 \ge 0$, $y_{r+1} \ge 0$, and $y_i > 0, i \in [2, r]$. Hence, the bit vector can be represented schematically as:

$$\underbrace{00...0}_{y_1}\underbrace{11...1}_{x_1}\underbrace{00...0}_{y_2}\underbrace{11...1}_{x_2}\cdots\underbrace{11...1}_{x_r}\underbrace{00...0}_{y_{r+1}}$$

Let $\bar{y}_1 = y_1 + 1$, $\bar{y}_{r+1} = y_{r+1} + 1$, and $\bar{y}_i = y_i, i \in [2, r]$, which satisfy:

$$\sum_{i=1}^{r+1} \bar{y}_i = n - m + 2. \tag{4}$$

Since each $\bar{y}_i$ is a positive integer, there are $\binom{n-m+1}{r}$ outcomes of (4), so does (3). Hence, the total number of vectors that exactly have $r$ runs of '1's is $\binom{n-m+1}{r}$, multiplied by the number of positive integral solutions of $\sum_{i=1}^{i=r} x_i = m$. Namely, there are totally $\binom{n-m+1}{r}\binom{m-1}{r-1}$ outcomes resulting in $r$ runs of '1's. The probability of getting an $r$ runs is:

$$\begin{aligned} P(\{r \text{ runs of '1's}\}) &= \frac{\binom{n-m+1}{r}\binom{m-1}{r-1}}{\binom{n}{m}} \\ &= \frac{\binom{n-m+1}{r}\binom{n-(n-m+1)}{m-r}}{\binom{n}{m}}. \end{aligned} \tag{5}$$

According to (5), clearly, the variable $r$ follows the hypergeometric distribution, i.e.,

$$r \sim H(n - m + 1, m, n). \tag{6}$$

By hypergeometric distribution, we get the expected value of $r$ (the proof can be found in [21]):

$$E(r) = \frac{(n-m+1)}{n}m. \tag{7}$$

Hence, tag bundling reduces the number of selects from $m$ to $\frac{(n-m+1)}{n}m$. When $m$ is large, e.g., close to $n$, the improvement is great. Besides, select combination also can reduce the number of selects by $w$ times, where $w$ is the number of masks supported by a reader in a select. Finally, we have the number $N_s$ of selects needed by WB:

$$Ns = \frac{E(r)}{w} = \frac{(n-m+1)}{n \times w}m. \tag{8}$$

According to (8), when $m = 0.5n$ and $w = 4$ (Alien readers and tags [19]), the number of selects is about $\frac{1}{8}m$, which reduces the selection overhead by 87.5%.

## IV. Generalized WB

The performance improvement of WB benefits from the use of bit vectors. Given the tag set $\Gamma$ with $n$ tags, each tag needs to allocate $2n$-bit buffer for storing the bit vector. This is fine when the tag set is small. For example, the tag chip of Impinj Monza 4QT [20] offers 512-bit user memory (MemBank-3) for customized use; up to 256 tags can be supported. As the tag population grows, WB may fail to afford so many tags. One way to gain some insight into this problem is to cut the length by half at the expense of communication overhead.

More specifically, each tag holds an $n$-bit vector rather than $2n$. For $t_i$, the $i$-th bit is set to '1' while other bits are set to '0'. To select a run of $\Gamma_{ij} = \{t_i, t_{i+1}, ..., t_j\}$, a select with two masks are needed. The first mask is from the $(j+1)$-th bit to the $n$-th bit, and the second is from the first bit to the $(i-1)$-th bit, $i \geq 2$ (if $i = 1$, the second mask is removed). To do so, the select settings are as follows:

$$\begin{aligned} &① \ \Gamma_{ij} \leftarrow AB: \ \mathcal{S}(2, 0, 3, j+1, n-j, 0), \\ &② \ \Gamma_{ij} \leftarrow A-: \ \mathcal{S}(2, 1, 3, 1, i-1, 0). \end{aligned} \tag{9}$$

By this means, the run of $\Gamma_{ij}$ can be selected by two masks, with the length of bit vector being cut by half. This is a trade-off between the memory space and the write efficiency. Although it can raise the limit of the number of tags to be accommodated, it is far away from a complete solution. To make WB more scalable to embrace the large RFID system, we take one step further to give a more generalized method, which not only fits the large tag set well but also hardly sacrifices the write efficiency.

The solution is to divide the available memory space into two parts: group region and vector region. The former is used to partition tags into different groups; the latter acts as the bit vector as is. Consider an $f$-bit memory, where the group region takes up $f_1$ bits and the vector region $f_2$ bits ($f = f_1 + f_2$). There are totally $2^{f_1}$ groups, each of which corresponds to an $f_2$-bit vector, which is able to deal with $\frac{f_2}{2}$ tags. Namely, $f_2 \times 2^{f_1-1}$ tags can be supported by an $f$-bit vector. For example, given a 110-bit memory where $f_1 = 10$ and $f_2 = 100$, the method is able to accommodate more than 50,000 tags, which is sufficient for most cases in practice.

Now we detail how to use these two regions. Consider the tag set $\Gamma$. Given $f_1$ and $f_2$, we first divide $\Gamma$ into several groups, each of which has $\frac{f_2}{2}$ tags. Assume the $i$-th group is $\Gamma^i = \{t_1^i, t_2^i, ..., t_{f_2/2}^i\}$. For the tag $t_c^i$, its group region is set to the value of $i$, and all bits of the vector region are initialized to 0 except for the $c$-th bit and the $(n+c)$-th bit (that are set to '1'), which is shown in Fig. 7.

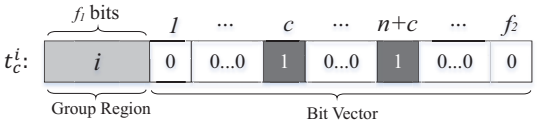



Fig. 7: The group ID and bit vector assigned to the $c$-th tag in the $i$-th group.

For the tag set $\Gamma'$ of interest, we select them group by group. Take the $i$-th group for example. We first obtain the subset of $\Gamma'$ that is classified into the $i$-th group, which is referred to as $\Gamma'^i$. According to $\Gamma'^i$, we get each run $\Gamma_j^i$ in the $i$-th group, where $\Gamma_j^i$ indicates the $j$-th run of the $i$-th group. By masking these runs, we can isolate $\Gamma'^i$ from others. To do so, an intuitive way is carrying out two mask settings each time: one for group ID and the other for a run. This works but suffers from repetitive settings on the same group ID, leading to waste of communication overhead. Instead, we can execute the mask on the group ID only once. The key is to place the setting of group ID at the last when dealing with a group. Namely, we mask all runs in a group first and the group ID is set after that. In this way, the redundancy can be avoided.

Alg. 1 shows the process of selecting $\Gamma'^i$, where $r_i$ is the number of runs in the group, $p_j^i$ and $l_j^i$ indicate the matching position of the run $\Gamma_j^i$. Line 1 sets the inventoried flags of the first run $\Gamma_1^i$ to $A$, while others are set to $B$. The action $AB$ serves as two purposes: one is to set $\Gamma_1^i$ to A; the other is to initialize other tags to $B$. After that, we progressively set the left runs in the $i$-th group $\Gamma^i$, which is shown in Lines $2 \sim 4$. The reason why $A-$ is adopted is that this action will set the matching runs accordingly but not change the previous settings of preceding runs. The result is that the tag subset $\Gamma'^i$ is chosen (inventoried flags are set to $A$). Besides $\Gamma'^i$, however, the same runs in other groups might also be mistakenly selected due to the same bit vectors. To remove these false positives, we need to carry out another select to determine the tags from only $\Gamma^i$, which is shown in Line 5. By the action of $-B$, all tags out of $\Gamma^i$ are set to $B$, while the matching tags ($\Gamma^i$) remain unchanged. Namely, only all runs in $\Gamma^i$ are selected (the inventoried flags are $A$); other groups with the same runs are removed. Compared with the basic WB, the generalized WB just increases the overhead of masking group IDs (Line 5 in Alg. 1). Since there are many runs in a group, this extra communication overhead that apportions to each run is negligible. In other words, by dividing the available memory space into two regions, WB is robust to large-scale RFID systems, at extremely low cost of the time efficiency.

To better clarify the enhanced WB, we use a scale-down

**Algorithm 1:** Selecting $\Gamma'^i$ in the $i$-th group.

**Input**: Each run $\Gamma^i_j$ in the $i$-th group.

$\Gamma^i_1 \leftarrow AB \ : \ \mathcal{S}(2, 0, 3, p^i_1, l^i_1, 0)$;
**for** $(j = 2; \quad j \leq r_i; \quad j++)$ **do**
  $\Gamma^i_j \leftarrow A- \ : \ \mathcal{S}(2, 1, 3, p^i_j, l^i_j, 0)$;
**end**
$\Gamma^i \leftarrow -B : \mathcal{S}(2, 2, 3, 1, f_1, i)$;

RFID system to illustrate the selection process. As shown in Fig. 8, the tag set $\Gamma$ consists of 12 tags $\{t_{1-12}\}$, the tag subset $\Gamma'$ to be written is $\{t_{1-3}, t_{5-6}, t_{8-10}, t_{12}\}$, and the available memory space is 16 bits long. Since the memory space is insufficient to accommodate 12 tags (8 tags at most), we divide the memory space into two parts: the first 4 bits are for the group ID and the left for the bit vector. By this means, there are two groups: $t_{1-6}$ are the first and $t_{7-12}$ are the second. The vector settings are the same as those in the basic WB. After that, the tags of interest are selected group by group. Take the first group for example. There are two runs $\Gamma^1_1 = \{t_{1-3}\}$ and $\Gamma^1_2 = \{t_{5-6}\}$. Three masks are needed based on Alg. 1:

$$\begin{aligned} &① \ \ t_{1-3} \leftarrow AB : \ \ \mathcal{S}(2, 0, 3, 8, 3, 000_2), \\ &② \ \ t_{5-6} \leftarrow A- : \ \ \mathcal{S}(2, 1, 3, 11, 4, 0000_2), \\ &③ \ \ \Gamma^1 \leftarrow -B \ : \ \ \mathcal{S}(2, 2, 3, 1, 4, 0001_2). \end{aligned} \tag{10}$$

The first mask sets the tags whose memory starting at the 8th bit and ending at the 10th bit is equal to $000_2$ to $A$, while the other not-matching tags are set to $B$. Therefore, $t_{1-3}$ and $t_{7-9}$ are selected. In the second step, a mask with action of $A-$ is executed, which makes $t_{5-6}$ and $t_{11-12}$ included. Since the action of $A-$ does not change the state of not-matching tags, the previous selected tags including $t_{1-3}$ are still $A$. After that, we remove the false positives (that do not belong to the first group $\Gamma^1$), i.e., $t_7$ and $t_{11}$, by executing the third mask $S(2, 2, 3, 1, 4, 0001_2)$. The action $-B$ does nothing on the matching tags (i.e., all tags in $\Gamma^1$) but sets the tags out of $\Gamma^i$ to $B$, such that only the runs in $\Gamma^1$ remain unchanged and there are no false positives caused by other groups. It is worth noting that although the tags $t_{8-9}$ and $t_{12}$ to be written are also removed by the third step, they will be selected shortly later when dealing with the second group. In this example, 16-bit memory is able to accommodate up to $2^4 \times 6 = 96$ tags (this example just uses 2 groups of 16), which is far superior to 8 tags compared with the basic WB.

$t_1$ 0001 | 100000100000
$t_2$ 0001 | 010000010000
$t_3$ 0001 | 001000001000
$t_4$ 0001 | 000100000100
$t_5$ 0001 | 000010000010
$t_6$ 0001 | 000001000001

(a) The first group.

$t_7$ 0010 | 100000100000
$t_8$ 0010 | 010000010000
$t_9$ 0010 | 001000001000
$t_{10}$ 0010 | 000100000100
$t_{11}$ 0010 | 000010000010
$t_{12}$ 0010 | 000001000001

(b) The second group.

Fig. 8: An illustration of generalized WB, where $\Gamma = \{t_{1-12}\}$, $\Gamma' = \{t_{1-3}, t_{5-6}, t_{8-10}, t_{12}\}$, and the available memory is 16 bits long: 4-bit group ID and 12-bit bit vector.


(a) Readers. (b) Experimental setup.

Fig. 9: Experimental Setup.

## V. Implementation & Evaluation

### A. Experimental Setup

We implement and evaluate WB using commodity RFID readers and tags. As shown in Fig. 9(a), six models of UHF RFID readers from three most experienced RFID suppliers are used in our experiments, including ThingMagic [22], Impinj [20], and Alien Inc. [19]. Each reader is connected to a directional antenna that is with 9 dBic gain and operates at around 920 MHz. To better mimic a real RFID system and extensively study the performance of WB in practice, we set up our experiments in a library, where up to 1000 commodity tags are used, which is shown in Fig. 9(b). We first perform a real test to investigate the certain capabilities concerned by this study. As shown in Table II, all readers support the select operation and the write operation. The setting of the inventoried flag is offered by some readers and tags, including M6E, Mercury6, ALN-F800, ALN-9900+, and all investigated tags. For the blastwrite operation, only Alien readers and tags are available. Therefore, they are taken as the vehicle to implement and evaluate WB in what follows. We assert that the disability of other readers and tags may be due to the fact that they support this function under the hood, but do not expose this level details to users.

### B. Number of Selects

In Section III-C, we have discussed the expected number of selects that is needed by WB. We now validate this theoretical analysis. As shown in Fig. 10, we fix $n$ to 1000 and vary $m$ from 100 to 900, where $n$ is the number of tags in $\Gamma$ and $m$ is the number of tags in $\Gamma'$. The dash line is the number of selects required by the exclusive write (EW), which is proportional to $m$ (each write corresponds to a select). By contrast, the solid line (WB-T) that indicates the theoretical number of selects needed by WB experiences a different trend. It first increases with the number $m$ of tags in $\Gamma'$. After peaking at the maximum when $m = 0.5n$, it drops over $m$. The reason is that, when $m$ is progressively close to $n$, the probability that multiple tags in $\Gamma'$ form a run increases, which accordingly leads to the reduction in the number of runs. Compared with EW, we observe that WB greatly reduces the number of selects, especially when $m$ is close to $n$. For example, when $m$ is 500, WB reduces the number of selects by 87.5%, from 500 to 62.6, which well indicates the good performance of WB. In

TABLE II: Capabilities of commodity readers and tags.

| Functions | Readers | | | | | | Tags | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ThingMagic | | Impinj | | Alien | | Impinj Monza | | | | | Alien Higgs™ | |
| | Mercury6 | M6e | R220 | R420 | F800 | 9900+ | 4 | 5 | R6 | R6-P | R6-C | 4 | EC |
| `Select` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| `Write` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Flag(A/B) | ✓ | ✓ | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| `BlastWrite` | × | × | × | × | ✓ | ✓ | × | × | × | × | × | ✓ | ✓ |

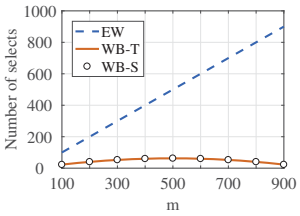



Fig. 10: Reduction in the number of selects.

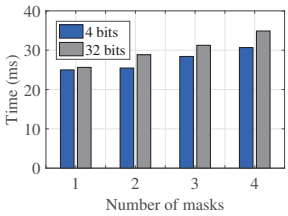
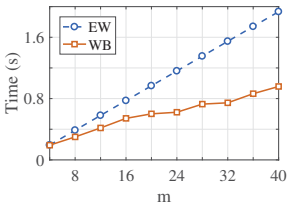
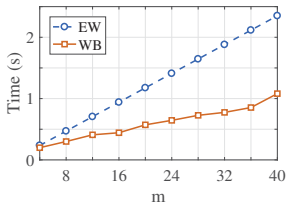



Fig. 11: Time duration of a select.




(a) 16-bit data.




(b) 32-bit data.

Fig. 12: Time efficiency of WB.

addition, the circle plots (WB-S) depict the simulation results of WB. Every plot is the average outcome of 100 runs, where $m$ tags of interest are randomly picked from $n$. As we can see, the tightness between the simulation value and theoretical value demonstrates that the expected value (see (8)) can well depict the actual number of selects in WB.

### C. Time Duration of a Select

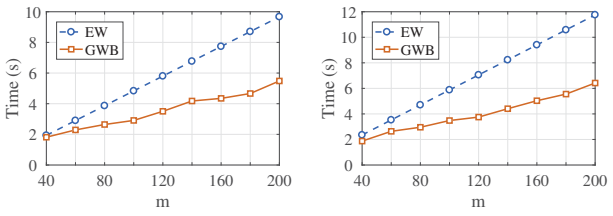
In WB, the approach of select combination is able to fill multiple masks in a select command. In this experiment, we study the duration of the select command with different numbers of masks. Since the Alien readers and tags [19] support up to four masks in a select, we vary the number of masks from 1 to 4. In addition, two kinds of masks are taken into account: one is with 4 bits long and the other is with 32 bits long. As shown in Fig. 11, the time duration of a select sees a slight increase over the number of masks and the length of masks. That is because, besides masks, a select command comprises many other overheads, such as the preamble, frame-sync, CRC, communication interval [15], which make mask settings take up a small portion of the transmission overhead of a select. Therefore, the reduction in the number of selects is close to that in the duration of all selection operations.

### D. Time Efficiency of WB

In Fig. 12, we compare the time efficiency of WB with that of exclusive write (EW). As the Alien Higgs™ tags (Table II) offer up to 128-bit user memory, we divide it into two parts. The first 32 bits are used for data storage (16-bit and 32-bit data are considered in the experiment). The left 96-bit memory is used for encoding, in which the first 16-bit is for the group region (for generalized WB) and the remaining 80-bit is for the bit vector. In the experiment, the number $n$ of tags in $\Gamma$ is set to $80/2 = 40$ tags (for more tags, generalized WB is needed, which will be evaluated shortly later), and the number $m$ of tags in $\Gamma'$ ranges from 4 to 40. As shown in Fig. 12(a) where the data is 16 bits long, both WB and EW see a rise trend over the number $m$ of tags in $\Gamma'$. That is intuitive because the inventory operation and the write operation are mandatory when updating each tag's memory. With the increase of $m$, the number of inventories and writes rises accordingly, leading to higher overall latency. The difference is that WB is more time efficient than EW due to the reduction in the number of selects. For example, when $m = 32$, WB drops the execution time from 1.54s to 0.73s, which produces a 2.1× write amplification in the ballparks. The similar conclusion can also be drawn in Fig. 12(b) where the data is 32 bits long. For example, when $m = 32$, the execution time is reduced by more than half, from 1.85s to 0.79s. These positive results clearly show that our protocol WB is able to improve the write throughput in commodity RFID systems.

### E. Time Efficiency of Generalized WB

Generalized WB (GWB) is to break through the barriers of small tag memory and make WB scalable to large RFID systems. Next, we evaluate the performance of GWB. As aforementioned, since the group region and the bit-vector region are set to 16 bits and 80 bits long respectively, the number of tags that can be accommodated by GWB is $2^{16} \times 80/2 = 2,621,440$, which can well fit most application scenarios. The number $n$ of tags is fixed to 200 and the tags of interest are produced randomly, varying from 40 to 200. Since there are 200 tags in total, 5 groups are needed, each of which owns 40 tags. By running Alg. 1 for each group, we can select the wanted tags in each group and conduct the blastwrite on the selected tag set for the purpose of group write. As shown in Fig. 13(a) where the data to be written is 16 bits long, GWB is still far superior to EW. For example, when $m = 160$, GWB drops the execution time from 7.9s to 4.1s, which produces a 1.9× write amplification approximately. Compared with the basic WB, the similar performance gain indicates that the extra cost (masking only 5 group IDs) introduced by GWB is almost negligible. In other words, GWB is able to generalize WB to large RFID systems, with a little side effect on the write performance. The similar conclusion can also be drawn in Fig. 13(b) where the data is 32 bits long.




(a) 16-bit data. (b) 32-bit data.

Fig. 13: Time efficiency of GWB.

## VI. Related Work

In RFID systems, most existing work studies the problem of improving the read throughout from tags to the reader, i.e., collecting the tag information in a time-efficient way. This information is either the basic tag ID or the functional data. For the ID information, many tag identification protocols are proposed, which generally fall into two categories: the ALOHA-based [23] and tree-based [24]. The former carries out a slotted frame to single out each tag for reporting tag IDs; the latter iteratively queries a subset of tags with a dynamic prefix of the tag IDs until a singleton tag is identified. After that, existing work has shifted to collect functional RFID data. For example, cardinality estimation [10], [11] is to count the number of tags in a large-scale RFID system; missing tag identification [13] aims to identify whether and which tags are absent; the searching problem [9] is to find a group of tags of interest from the existing tag set.

In recent years, a little work starts to investigate the problem of improving the write throughout in RFID systems. Liu et al. [16] are the first to formulate the grouping problem and design an efficient concurrent grouping protocol to make the multicast possible. Zhu et al. [17] study the grouping problem in a dynamic scenario and propose a binary grouping protocol that improves the grouping efficiency by flipping the corresponding bits of the group IDs. Yu et al. [18] employ multiple seeds to build a composite indicator vector that indicates the assigned seed in each slot, which reduces transmissions of useless information and thereby improves time efficiency. Although some significant advancement has been achieved, the existing solutions to the problem of improving write throughput are out of the scope of the C1G2 standard and cannot be deployed on commodity RFID systems.

## VII. Conclusion

In this work, we investigate the group write problem and make a fresh attempt to improve the write throughput in commodity RFID systems. By real experiments, we observe that the select operation takes up high overhead of a write cycle and the reduction in the number of selects is the key to amplify the write throughput. Following this idea, we design an efficient write bundling (WB) scheme that improves the write performance with the technologies of tag bundling and select combination. We implement WB in a commodity RFID system; there is no need of any software modifications or hardware augment. Extensive experiments show that WB is able to produce a $2\times$ write amplification in practice.

## Acknowledgment

This work is supported by the National Natural Science Foundation of China (61702257, 61771236, 61772251), Natural Science Foundation of Jiangsu Province (BK20170648), Jiangsu Key R&D Plan (Industry Foresight and Common Key Technology, BE2017154), Project funded by China Postdoctoral Science Foundation, Fundamental Research Funds for the Central Universities, and Collaborative Innovation Center of Novel Software Technology and Industrialization. Jia Liu and Lijun Chen are corresponding authors.

## References

[1] J. Han, C. Qian, X. Wang, D. Ma, J. Zhao, P. Zhang, W. Xi, and Z. Jiang, "Twins: Device-free object tracking using passive tags," in *Proc. of IEEE INFOCOM*, 2014, pp. 469–476.
[2] G. Wang, C. Qian, L. Shangguan, H. Ding, J. Han, N. Yang, W. Xi, and J. Zhao, "HMRL: Relative localization of RFID tags with static devices," in *Proc. of IEEE SECON*, 2017, pp. 1–9.
[3] 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.
[4] X. Liu, J. Yin, S. Zhang, B. Ding, S. Guo, and K. Wang, "Range-based localization for sparse 3D sensor networks," *IEEE Internet of Things Journal, DOI:10.1109/JIOT.2018.2856267*, 2018.
[5] Y. Bu, L. Xie, Y. Gong, C. Wang, L. Yang, J. Liu, and S. Lu, "RF-Dial: An RFID-based 2D human-computer interaction via tag array," in *Proc. of IEEE INFOCOM*, 2018, pp. 837–845.
[6] J. Liu, F. Zhu, Y. Wang, X. Wang, Q. Pan, and L. Chen, "RF-Scanner: Shelf scanning with robot-assisted RFID systems," in *Proc. of IEEE INFOCOM*, 2017, pp. 1–9.
[7] C. Wang, L. Xie, W. Wang, T. Xue, and S. Lu, "Moving tag detection via physical layer analysis for large-scale RFID systems," in *Proc. of IEEE INFOCOM*, 2016, pp. 1–9.
[8] X. Liu, J. Cao, K. Li, J. Liu, and X. Xie, "Range queries for sensor-augmented RFID systems," in *Proc. of IEEE INFOCOM*, 2018, pp. 1–9.
[9] Y. Zheng and M. Li, "Fast tag searching protocol for large-scale RFID systems," in *Proc. of IEEE ICNP*, 2011, pp. 363–372.
[10] C. Qian, H. Ngan, Y. Liu, and L. Ni, "Cardinality estimation for large-scale RFID systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 9, pp. 1441–1454, 2011.
[11] M. Shahzad and A. X. Liu, "Every bit counts: Fast and scalable RFID estimation," in *Proc. of ACM MobiCom*, 2012, pp. 365–376.
[12] 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.
[13] J. Yu, W. Gong, J. Liu, L. Chen, K. Wang, and R. Zhang, "Missing tag identification in COTS RFID systems: Bridging the gap between theory and practice," *IEEE Transactions on Mobile Computing, DOI:10.1109/TMC.2018.2889068*, 2018.
[14] S. Qi, Y. Zheng, M. Li, Y. Liu, and J. Qiu, "Scalable data access control in RFID-enabled supply chain," in *Proc. of IEEE ICNP*, 2014, pp. 71–82.
[15] "EPC radio-frequency identity protocols generation-2 UHF RFID standard," GS1, ISO/IEC 18000-63, 2018.
[16] J. Liu, B. Xiao, S. Chen, F. Zhu, and L. Chen, "Fast RFID grouping protocols," in *Proc. of IEEE INFOCOM*, 2015, pp. 1948–1956.
[17] F. Zhu, B. Xiao, J. Liu, Y. Wang, and L. j. Chen, "Dynamic grouping in RFID systems," in *Proc. of IEEE SECON*, 2017, pp. 1–9.
[18] J. Yu, J. Liu, L. Chen, and Y. Zhu, "Efficient group labeling for multi-group RFID systems," in *Proc. of IEEE/ACM IWQoS*, 2017, pp. 1–2.
[19] "Alien Technology," http://www.alientechnology.com.
[20] "Impinj Inc." http://www.impinj.com.
[21] S. Ross, *A First Course in Probability*. Pearson Prentice Hall, 2010.
[22] "ThingMagic," http://www.thingmagic.com.
[23] 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.
[24] L. Pan and H. Wu, "Smart trend-traversal: A low delay and energy tag arbitration protocol for large RFID systems," in *Proc. of IEEE INFOCOM*, 2009, pp. 2571–2575.