Time-efficient Range Detection in Commodity RFID Systems

Jia Liu, Xingyu Chen, Haisong Liu, Hualin Gong, Yanyan Wang, Lijun Chen
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

***Abstract*—RFID is becoming ubiquitously available in our daily life. After RFID tags are deployed to make attached objects identifiable, a natural next step is to communicate with the tags and collect their information for the purpose of tracking tagged objects or monitoring their surroundings in real time. In this paper, we study an under-investigated problem *range detection* in a commodity RFID system, which aims to check if there are any tags with the data between an upper and lower boundary in a time-efficient way. This is important especially in a large RFID system, which can help users quickly pinpoint the target tags (if any) and give an early warning to users for taking urgent actions and reducing the potential risk in the nascent stage. We propose two tailored protocols, selective query and range query (RQ), to achieve range detection within the scope of the C1G2 standard. The novelty is that, instead of querying each tag, we exploit the capability of C1G2-compatible selection and quickly separate target tags from others by silencing most of tags. The final result is that our best protocol RQ is able to achieve a range detection with only one query command. We implement the proposed protocols in commodity RFID systems, without any modifications of hardware. Extensive experiments show that RQ is able to improve the time efficiency by near 30×, compared with the baseline.**

***Index Terms*—RFID, Range detection, C1G2, Time efficiency**

I. Introduction

Radio frequency identification (RFID) is becoming increasingly ubiquitous in a variety of applications, including library inventory [1], [2], object tracking [3]–[6], warehouse management [7]–[9], etc. After RFID tags are deployed to make the attached objects identifiable, a natural next step is to communicate with the tags and collect their information of interest. This information can be some static data that are preloaded in the tag's memory for reflecting the tagged product's attributes (e.g., expiry date) or some dynamic sensing data (e.g., temperature) in a sensor-augmented RFID system for item-level monitoring. By collecting this tag information, we can either track the status of tagged objects or monitor their surroundings in real-time.

In this paper, we study an under-investigated problem *range detection* in a commodity RFID system, which aims to check whether there are any tags with the data between an upper and lower boundary. These tags are referred to as *target tags*. High time efficiency is important especially in a large RFID system, which can help users quickly pinpoint the target tags (if any) and give an early warning to them for taking urgent actions and reducing the potential risk in the nascent stage. For example, in a supermarket, the staff may need to do the live query for checking whether there are expired goods by collecting the sell-by date from tags. Consider a chilled food storage chamber (or a library), where each food (or each bookshelf) is affixed with a sensor-augmented tag equipped with a thermal sensor. If the temperature data of a tag is higher than a threshold, an advance warning (for food to be spoiled or for fire) should be activated to protect people and assets.

An intuitive solution to range detection is individually querying each tag and checking whether it holds data within the given range, one at a time. This method is foolproof but suffers from long time period in a large scale RFID system. In some cases the number of target tags is far less than other tags, it is a waste of time to do the inventory on the entire tag set. In recent years, some advanced work has been proposed to collect tag information in an efficient way [10]–[15]. For example, Chen et al. [11] design a multi-hash method to avoid tag collision and improve the time efficiency. Yue et al. [12] study the multi-reader RFID system and use a distributive Bloom filter to do the tag inventory quickly. Liu et al. [15] design an incremental polling protocol that sharply drops the length of the polling vector from 96 bits to less than 2 bits, which saves the polling overhead for information collection. In spite of the advancement, existing work is incompatible with EPCglobal Class1 Gen2 (C1G2) [16], which is the worldwide UHF RFID standard. Namely, none of them can be implemented in a commodity RFID system.

In light of this, we explore the full potential of C1G2 and deliver two tailored solutions that drive significant improvement to range detection in commodity RFID systems. The basic idea is that, instead of checking each tag, we first silence the majority of tags and later pinpoint target ones (if any) as soon as possible. We find the select command specified by C1G2 is potential to help us reach this goal. It allows a reader to choose a subset of tags that participate in the subsequent query. With this ability, we come up with our first solution called *selective query* (SQ). SQ partitions the tag populations into several groups according to the value of the data. Rather than doing individual inventory, SQ queries only one tag in each group and silences the remaining tags in the same group with the select command. In this way, the number of queries is proportional to only the number of groups, which is a sharp decline compared with the individual query.

In spite of the improvement, it is still a waste of time to collect each group. This motivates us to take one step further: can we do the tag query only once, regardless of the number of groups or the number of tags? The answer is yes. We

978-1-7281-6992-7/20/$31.00 ©2020 IEEE

propose our second protocol called *range query* (RQ). RQ first separates all target tags (if any) from others through several select commands and later does the tag inventory over the selected tag subset. If there is any reply, a target tag is detected. The key challenge is how to separate the target tags from the entire tag set without any prior knowledge of the tag IDs. We propose two approaches substring masking and mask combination that jointly minimize the number of selects. The final result is that RQ needs only a few selects (e.g., two in most cases) together with a single tag inventory to detect the target tag, regardless of the number of tags. The main contributions of this paper are three-fold.

- We study the under-investigated problem range detection in a commodity RFID system, which helps quickly pinpoint target tags and reduce the potential risk in the nascent stage.
- We explore the full potential of C1G2-compatible commands and deliver two tailored solutions. By jointly using the technologies of substring masking and mask combination, we are able to detect the target tags with only one inventory, regardless of the number of tags.
- We implement the proposed protocols in a commodity RFID system with 1000 tags. Extensive experiments show that our best protocol RQ can sharply reduce the detection time from 9.6s to only 0.33s, improving the time efficiency by near 30×, compared with the baseline.

The rest of the paper is organized as follows. Section II formulates the problem of range detection. Section III shows an intuitive solution: exclusive collection. Section IV presents the method of selective query. Section V details the approach of range query. Section VI evaluates the protocol performance. Section VII discusses the related work. Finally, Section VIII concludes this paper.

II. Problem Formulation

We consider an RFID system that consists of a reader and a number of tags. Each tag has a unique tag ID that exclusively indicates the associated object. By communicating with the tags and collecting some specific information from these tags, the reader is able to grasp the attributes of the tagged objects (e.g., expiry date) or the status of the surroundings (e.g., temperature and humidity). *The problem of range detection is to check if there are any tags that hold data with the value between an upper and lower boundary.* These tags are referred to as target tags. More specifically, consider a tag set $\Gamma = \{t_1, t_2, ..., t_n\}$, in which each tag t_i, $1 \leq i \leq n$, holds data d_i of interest. We assume that the value of d_i is a positive integer, each of which represents an attribute value or a specific sensor value. For example, we can define the expiry date of 2020 as follows: 1 indicates 'Jan-1-2020', 2 indicates 'Jan-2-2020', ..., and 366 indicates 'Dec-31-2020'. The mapping rule and the range of interest depend on the applications, which are out of the scope of this paper. It is difficult (time-consuming) to figure out whether target tags exist in a large RFID system. *Our objective is to minimize the detection time with only the commodity RFID device; any modifications to the MAC-layer communication protocol or hardware are not allowed.* The high time efficiency is important especially in a large RFID system, which can give an early warning to users and help them reduce the potential risk in the nascent stage.

III. Exclusive Collection

The Class1 Gen2 (C1G2) protocol [16] is a worldwide UHF RFID standard that defines the physical interactions and logical operations between the commodity readers and tags. According to C1G2, the RFID reader collects tag information through an inventory frame, which consists of a group of time slots. The time slot is a short time window, within which a tag can communicate with the reader. Each tag randomly picks one of these time slots and transmits its tag ID together with the user data (if required) to the reader in that slot. According to the tag's choice, the time slots fall into three categories: singleton slots picked by exactly one tag, collision slots chosen by more than one tag, and empty slots with none of tags chosen. Only the singleton slot is available to collect tag information, since the empty slot has no tags and the collision slot suffers from tag collision.

To do range detection within the scope of C1G2, we can execute the inventory on each tag of Γ. Once we get the data d_i from the tag t_i, we check whether it is within a given range. If yes, we report the event. Otherwise, we continue to do the next inventory for another tag. This solution is intuitive and foolproof. However, it is time-consuming since C1G2 allows only one tag to be read at a time; there are near n reads needed for range detection. In some cases, the number of target tags is far less than that of all tags (e.g., 1000 tags with only one target). It is a waste of time to do the inventory on the entire tag set. Reducing the number of inventories and quickly pinpointing the target tags (if any) amongst the large tag populations is the key to performance boost.

IV. Selective Query

A. Basic Idea

The low time efficiency of exclusive collection is due to the fact that the reader has to communicate with each tag, even though the tag is not the target. If we can silence most of these tags, a great deal of communication overhead can be avoided. Let us take a closer look at the tagged objects in practice. We get two observations. First, in some cases the number of values in the domain of $\{d_i\}$ to be collected is far less than the number of tags. For example, consider the expiry date in 2020. The number of different values in the domain is 366, which is much smaller than the number of goods in a supermarket (e.g., Wal-Mart offers 142,000 different items on average in its supercenters [17]). According to pigeonhole principle, there must be some tags holding the same data. Second, different tags may have the same attribute or similar surroundings. For example, the expiry date of the same batch of milk is usually identical; the nearby sensor-augmented tags

are likely to report the same temperature.[1] These two findings shed light on the basic idea of our protocol: *collect the same data only once and silence the left tags having the same data.* In this way, the number of reads is equal to the number of different values amongst the tags (which is smaller than that in the domain of d_i sometimes); many tags with the same data will not participate in the subsequent query, greatly saving the communication time. Next, we discuss how to silence the tags and pinpoint the target tags quickly within the scope of the C1G2 standard, which is referred to as *selective query* (SQ).

B. Selective Query (SQ)

C1G2 specifies a select operation before the inventory, which allows a reader to choose a specific subset $\Gamma' \subseteq \Gamma$ of tags that participate in the incoming query. In other words, it is capable of silencing the tag set $\Gamma - \Gamma'$. Below, we first describe how a select operation works and then discuss how to use selective query to do range detection.

1) Select Function: The command `Select` determines which tags keep active by setting their inventoried flags, which 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, which could be one of the four banks: `MemBank-0`, `MemBank-1`, `MemBank-2`, and `MemBank-3`. The customized data are stored in `MemBank-3`, so the `MemBank` is always set to 3 in this work. `Pointer` indicates the starting position in the chosen memory bank. `Length` is the length of `Mask` that is a specific bit string determined by the application demands. If `Mask` is the same as the string that begins at `Pointer` and ends `Length` bits later in the memory of `MemBank`, the tag is matching. Otherwise, the tag is not-matching.

• `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 shown in Table II, there are eight actions, where matching and not-matching tags assert or deassert 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. Note that 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 (see [16]). In this paper, we take the inventoried flag in session 2 as the metric to show our protocol design. The other flags can be used in the similar way.

2) Query Function: Besides `Select`, C1G2 defines another common command called `Query`, which initiates and specifies an inventory frame over the active tag populations with a specific flag (A or B). `Query` includes three fields that we concern: `Sel`, `Session`, and `Target`. `Sel` consists of two bits that determine which tags respond to the reader: 00_2 and 01_2 indicate all matching tags by the previous `Select`; 10_2 indicates tags with deasserted SL flag ($\sim$SL); and 11_2 indicates tags with asserted SL flag (SL). Since SL is not used in this paper, this field is set to 0 all the time. `Session` selects a session for this inventory frame. As aforementioned, session 2 (S2) is used in this paper, so the value is set to 2 (10_2). `Target` determines which tags participate in the upcoming inventory frame: 0 indicates the tags with the flag A and 1 indicates B. Tags will invert their inventoried flags from A to B (or vice versa) after being successfully queried.

TABLE I: Select command.

	Command	Target	Action	MemBank	Pointer	Length	Mask
# of bits	4	3	3	2	EBV	8	Variable
description	1010	000: Inventoried (S0) 001: Inventoried (S1) 010: Inventoried (S2) 011: Inventoried (S3) 100: SL 101: RFU 110: RFU 111: RFU	See Table 2	00: FileType 01: EPC 10: TID 11: File_0	Starting Mask address	Mask Length (bits)	Mask value

3) Design of SQ: Given a tag set Γ and its data set $\{d_i\}$, we divide the tag populations into different groups according to the value of d_i. For any two tags t_i and t_j, if and only if the data d_i is identical to d_j, these two tags belong to the same group. Assume that there are k groups, which are $G = \{g_1, g_2, ..., g_k\}$. The objective of SQ is to query each group only once; multiple replies from the same group are a waste of communication overhead. Note that, no one knows the group information in advance. It is just defined for ease of presentation. We need to collect each group's data without any knowledge of the groups or tag IDs a priori.

In general, SQ consists of k rounds at most. In the first round, the reader inventories a tag t_i (the first active tag to be read) for collecting its data d_i. If d_i is within the given range, a target tag is detected. Otherwise, the reader uses the data d_i to do the mask to silence the tags in g_1', where g_1' is the group in which the tag t_i resides[2]. In this way, we have successfully collected the data of the tag set of the group g_1', with only a query together with a select command. The reader does the similar operations in the following rounds. The difference is that when silencing the group g_i' in the i-th round, we need to pay attention to the silenced tags in the previous rounds: the select operation issued in the i-th round is not supposed to activate these silenced tags and let them participate in the subsequent read cycles again.

Next, we detail how each round works with Gen2-compatible commands `Select` and `Query`. A `Select` command is denoted by:

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

[1]The case that does not follow the above two observations will be discussed in Section V.

[2]The reason for using g_1' instead of g_1 is to show that the order of groups to be queried in the rounds does not have to follow the order defined in G.

with the fields of `Target` (t), `Action` (a), `MemBank` (b), `Pointer` (p), `Length` (l), and `Mask` (m). Assume that the active tags are the ones with the flag A, while silenced tags are with the flag B. Initially, the reader broadcasts a select command to activate all tags by setting their flags to A. The `Select` command is as follows:

$$\texttt{Flag} \leftarrow AB: \ \mathcal{S}(2,0,3,0,0,0), \tag{2}$$

where $t = 2$ (010_2) means the operating object is the inventoried flag in session 2 (S2), $a = 0$ indicates that the inventoried flags of matching tags will be set to A while those of not-matching will be set to B (abbr. AB), $(p, l, m) = (0, 0, 0)$ means all tags within the reader's coverage zone are matching. By this means, all tags' flags are set to A. After that, the reader executes each round for collecting a group's data. It consists two operations: a `Query` and a `Select`. The `Query` is to do inventory on active tags, which is denoted by:

$$Q(\texttt{Sel}, \texttt{Session}, \texttt{Target}). \tag{3}$$

To query the active tags with inventoried flags in S2 being A, the query command shall be $Q(0, 2, 0)$. After this command is issued, an inventory frame is carried out. Each active tag (with the inventoried flag being A) randomly chooses a slot and replies to the reader in that slot. The reader is able to collect a tag's data in a singleton slot. Assume that the inventoried tag is t' and its reported data is d'. If d' falls into the given range, a target tag is detected. Otherwise, the reader needs to silence the group where the tag t' resides by issuing another `Select` command:

$$\texttt{Flag} \leftarrow B-: \ \mathcal{S}(2,5,3,p',l',d'), \tag{4}$$

where p' and l' indicate the matching position and the length of the data d', $a = 5$ means the matching tags are set to B while the not-matching tags remain unchanged. In other word, this command sets all tags with the data d' to B, while other tags do not change their flags.

Now we put all the pieces together and sketch the SQ protocol for range detection within the scope of C1G2. As shown in Alg. 1, Line 1 is for the reader to broadcast a `Select` to activate all tags in the field of view (the inventoried flags are set to A). Lines 2-10 detail the process of selective query, round by round. Each loop corresponds to a round. In each round, the reader first does the inventory on the active tags (Line 3). Assume the collected data is d'. If d' falls into the given range, a target tag is detected and the detection process ends (Lines 5-6). Otherwise, the reader needs to silence all tags that hold the same data d' by broadcasting a `Select` with the action $B-$ (Lines 7-8). The reader repeats the above process until all active tags are silenced or a target tag is detected.

TABLE II: 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

Algorithm 1: Selective Query.

```
1  Flag ← AB : S(2,0,3,0,0,0);
2  while there are still active tags (flag is A) do
3  |   Q(0,2,0);
4  |   Assume the collected data is d';
5  |   if d' is within the given range then
6  |   |   A target tag is detected; break;
7  |   else
8  |   |   Flag ← B− : S(2,5,3,p',l',d');
9  |   end
10 end
```

C. Performance Analysis

Now we discuss the execution time of SQ. Assume that the tag set $\Gamma = \{t_1, t_2, ..., t_n\}$ is partitioned into k groups according to the value of $\{d_i\}$. According to Alg. 1, the reader first broadcasts a `Select` to activate all tags in the field of view. After that, the reader deals with each group at a time: isolating a tag in a group, collecting its data, and silencing the whole group with the same data. If there is no target tag, the reader needs to go through all k rounds. The execution time T^* of SQ is:

$$T^* = t_s + k(t_q + t_s), \tag{5}$$

where k is the number of rounds, t_s and t_q are the time intervals of a `Select` command, an inventory frame issued by a `Query` command, respectively. In a more generalized case, if there exist some target tags, it is unnecessary for the reader to run k rounds. Once a target tag is detected, the reader can terminate the protocol immediately. If the protocol ends at the i-th round, it means that there are no target tags in the previous $(i-1)$ rounds and a target tag is detected in the i-th round. Let ρ be ratio of the number of target tags to n. The probability f_i that the reader detects a target tag or terminates the protocol in the i-th round is:

$$f_i = \begin{cases} (1-\mu_i)^{i-1}\mu_i, & i < k \\ (1-\mu_{k-1})^{k-1}, & i = k, \end{cases} \tag{6}$$

where μ_i indicates the probability that a target tag is detected in the i-th round, which is

$$\mu_i = \frac{\rho \times n}{n - \sum_{j=1}^{i-1} |g'_j|}, \tag{7}$$

where g'_j is the group to be collected in the i-th round, $|g'_j|$ is the number of tags in the group g'_j. The expression $(1-\mu_i)^{i-1}$ means the probability that the previous $(i-1)$ rounds do not have any target tags. If there are no any target tags in the previous $(k-1)$ rounds, the protocol executes k round for sure.

Thus, in the case of $i = k$, the probability is $(1 - \mu_{k-1})^{k-1}$ instead of $(1 - \mu_{k-1})^{k-1}\mu_k$. Finally, we have the execution time T of the protocol SQ:

$$T = t_s + (t_q + t_s)(\sum_{i=1}^{k} i \times f_i). \tag{8}$$

Clearly, Eq. (5) is just a specific case of Eq. (8), in which the number of target tags is equal to 0. According to Eq. (8), we can see that the execution time of SQ is proportional to the number k of groups, rather than the large number n of tags. In practice, different tags may have the same attribute or similar surroundings. For example, the production dates of the same batch of goods are usually identical; the nearby sensor-augmented tags are likely to report the same humidity. Assume that each group has 50 tags on average, i.e., $k = 0.02n$. It is potential for SQ to improve the time efficiency by an order of magnitude, compared with the method of exclusive collection. The performance evaluation can be seen in Section VI.

V. Range Query

The performance improvement of SQ benefits from the effective use of the select command together with the query command. By silencing many tags with the same data, most data collection can be avoided. However, SQ still suffers from two limitations. First, the objective of range detection just needs a binary answer: yes or no. It is a great waste of time for SQ to collect each group's data individually. Second, SQ performs well in the common cases that each group has many tags (i.e., $k << n$). However, in some specific cases when k is close to n, the number of rounds approaches to n, such that SQ degrades to the basic exclusive collection. Can we reduce the number of query rounds to only once, regardless of the number k of groups or the number n of tags? The answer is positive. In this section, we propose another protocol called *range query (RQ)* that further improves the time efficiency of range detection.

A. Design of RQ

In database, the range query is a common operation that retrieves all records where some value is between an upper and lower boundary. Similarly, in this paper, the range query answers three questions: i) are there some data in $\{d_i\}$ not greater than a lower boundary (this query is referred to as LRQ); ii) are there some data in $\{d_i\}$ greater than an upper boundary (URQ); iii) are there some data between an upper and lower boundary (LURQ). Below, we first detail LRQ. The other two queries can be easily generalized from LRQ, which will be discussed shortly later in Section V-B.

Assume the lower boundary is τ. The objective of LRQ is to detect whether there are any tags that hold the data d_i less than or equal to τ. The basic idea is separating all target tags (if any) from others first, and later carrying out the inventory frame over the selected tags. If there are any replies, a target tag is detected. Otherwise, there is no target tag. The key challenge is how to separate the target tags from the entire tag set without any prior knowledge of tag IDs.

As aforementioned, the select command allows a reader to choose a specific subset of tags that participate in the subsequent query operations, which is potential to do the tag partition. To do so, we need to activate the tags (if any) with the data no greater than τ while silencing other tags. An intuitive way is to check each value $d' \in [1, \tau]$ individually, one at a time. Specifically, for the first case of $d' = 1$, the reader issues a `Select` command to all tags, with the mask string 1 together with the action AB. After that, the tags with the data value equal to 1 are matching and their inventoried flags are set to A. On the contrary, the inventoried flags of other not-matching tags are set to B (see `Action` in Table II). The `Select` command is as follows:

$$\texttt{Flag} \leftarrow AB: \ \mathcal{S}(2, a = 0, 3, p', l', m = 1), \tag{9}$$

where $a = 0$ indicates that the inventoried flags of matching tags will be set to A while those of not-matching will be set to B, $(3, p', l')$ determines the matching position of the data. If and only if the data is same as $m = 1$, the tag is matching. Otherwise, the tag is not-matching. With this command, the tags with data equal to 1 are activated (the flag is A) and other tags are silenced. Instead of doing the tedious inventory immediately, we move to the next number. For the number i ($\leq \tau$), we conduct the similar select operation. The only difference is that the action is $A-$ and the mask value is i. The action $A-$ means that the inventoried flags of matching tags are set to A, whereas the other tags keep unchanged. In this way, we increasingly activate the tags with the data i, without changing the flags of existing tags. The `Select` command is as follows:

$$\texttt{Flag} \leftarrow A-: \ \mathcal{S}(2, a = 1, 3, p', l', i), \tag{10}$$

where $a = 1$ means the action of $A-$. By repeating the selects τ times, we are able to activate all tags (whose flags are A) with the data belonging to the interval $[1, \tau]$, and silence the remaining tags (whose flags are B). The next is to query the active tags (if any) with a single `Query` command:

$$\texttt{Query} - A: \ Q(0, 2, \texttt{Taget} = 0), \tag{11}$$

where $\texttt{Taget} = 0$ means doing the inventory on the tags with the flags being A, i.e., those with data belonging to the interval $[1, \tau]$. If any response is received by the reader, there must be some target tags for sure. By this means, LRQ sharply reduces the number of inventory rounds from k to exact one, which greatly saves the communication overhead, compared with selective query (SQ). However, this is not free. LRQ takes τ select operations to activate the wanted tags, which is a high cost when τ is large. For example, consider 10,000 tags and $\tau = 5000$. For SQ, the reader needs to conduct 100 select operations and 100 inventories (if each group has 100 tags on average). In contrast, LRQ takes 5000 selects together with one inventory to achieve the same task, which is time-consuming from a global view. The main reason is that LRQ deals with each number at a time. To improve the time efficiency, we need to operate multiple numbers in parallel.

To do so, the basic idea is that, instead of taking the specific value of a number as the mask, we seek for a common binary substring that is able to choose multiple numbers and thereby activate the potential target tags associated with these numbers, with only a single select command. More specifically, assume that the length of the number τ in binary representation is l'. Hence, the domain of the data of interest is $[1, 2^{l'}]$. Consider a specific case that $\tau = 2^x - 1$, where $x < l'$. Its binary representation is:

$$B(\tau = 2^x - 1): \underbrace{0\ 0\ 0\ 0\ 0}_{l'-x}\ \underbrace{1\ 1\ ...1\ 1\ 1}_{x} \tag{12}$$

Clearly, the rightmost x bits of the binary number in Eq. (12) are all 1s and the left $(l' - x)$ bits are all 0s. For any number larger than $2^x - 1$, the rightmost x digits start over, and next digit is incremented. In other words, the leftmost $l'-x$ bits are non-zero. In contrast, for all numbers that are smaller than $2^x - 1$, the leftmost $l' - x$ bits must be zeros (if not, this number is larger than $2^x - 1$ for sure). This observation enables LRQ to select all numbers in the interval $[1, \tau]$ via only a mask, i.e., the leftmost $l' - x$ bits. The `Select` is:

$$\texttt{Flag} \leftarrow AB: \ \mathcal{S}(2, a = 0, 3, p', l = l' - x, m = 0), \tag{13}$$

where $a = 0$ means the action of AB, $l = l' - x$ indicates that the mask is the leftmost $l' - x$ bits, and $m = 0$ means that mask string is zeros. This single select command is able to activate all tags with their data smaller than or equal to $\tau = 2^x - 1$ by setting their inventoried flags to A, regardless of the value of x. This is a great performance boost, compared with one-by-one selects. For example, when x is 10, the basic LRQ needs to execute the selects 1023 times, in comparison to only once by the improved LRQ.

However, things are not quite that simple, especially in a generalized case where τ can be any number in the domain. Consider another case that $\tau = 2^x$, where $x < l'$. There is no common mask that can choose all numbers in $[1, 2^x]$ with only one select command. To address this problem, we need to combine multiple selects together to incrementally activate tags. Namely, we first use the select command in (13) to choose all numbers in $[1, 2^x - 1]$. After that, we issue another select with the action $A-$ and the mask 2^x to incrementally choose the number 2^x:

$$\texttt{Flag} \leftarrow A-: \ \mathcal{S}(2, a = 1, 3, p', l', m = 2^x), \tag{14}$$

where the action $a = 1$ means $A-$ and the value of 2^x is the mask for activating the tags with the data being 2^x. Now the question is: given any τ in $[1, 2^{l'}]$, how many selects does LRQ need and what exactly are these selects in a generalized case. The answer is taking advantage of the specific case of $2^x - 1$, as many as possible. That is because all numbers in $[1, 2^x - 1]$ can be chosen by only one select command. Given any τ, we first discuss the case that τ is an odd number. Its binary representation $B(\tau)$ can be expressed as follows:

$$B(\tau): \ 0...0\ \underset{r_1}{\overset{\uparrow}{1}}\ 0...0\ \underset{r_2}{\overset{\uparrow}{1}}\ 0...0\ \underbrace{1\ 1...1}_{d(\tau)} \tag{15}$$

where $d(\tau)$ is the number of consecutive rightmost 1s. Let $\mathcal{R}(\cdot) = \{r_i\}$ be the index set of 1s in $B(\tau)$, where the index counts from right to left and the index of the rightmost bit is 0. Without loss of generality, we sort the indices and make them satisfy $r_i > r_j$ when $i < j$. For example, given a number $43 = 00101011_2$, we have $\mathcal{R}(00101011_2) = \{r_1 = 5, r_2 = 3, r_3 = 1, r_4 = 0\}$.

The basic idea of improved LRQ is to find the biggest number $2^x - 1$ that is smaller than τ. The reason is that it is the biggest sub-interval of $[1, \tau]$ that can be chosen by only one select command. For example, given 43, the biggest number $2^x - 1$ is 31. We can first mask the sub-interval $[1, 31]$ with the `Select` in (13). With the index set $\mathcal{R}(\tau)$, it is easy for us to find the biggest x, which is equal to r_1, i.e., $2^5 - 1 = 31$. The mask string is the left bits of the r_1-th bit together with another '0'. In this example, the left of the r_1-th bit are two zeros; the final mask string is three zeros '000'.

After that, we can repeat this process recursively. The difference is that, in the i-th select $(i > 1)$, we mask the numbers in $[2^{r_{i-1}}, 2^{r_{i-1}} + 2^{r_i} - 1]$ with the action $A-$. For example, in the second select of 43, we can take the left four bits of the r_2-th bit together with a '0' (i.e., '00100') as the mask string for select. The numbers in $[2^5, 2^5 + 2^3 - 1] = [32, 39]$ will be matching by this mask string, while others are not-matching. Finally, for the last $d(\tau)$ rightmost 1s, instead of checking each r_i, only one mask is able to deal with the left numbers. The mask string is the bits from the leftmost one to the $d(\tau)$-th bit (including the $d(\tau)$-th bit). For example, the mask string of the third selects for 43 is '001010'. It can select the numbers in $[40, 43]$. In this way, the numbers in the interval $[1, 43]$ are all selected and the tags (if any) holding any one of these numbers are activated (their flags are set to A). As we can see, only three selects are issued, which are listed below:

$$\begin{aligned}
&① \ [\ 1, 31] \leftarrow AB: \ \ \mathcal{S}(2, 0, 3, p', 3, m = 000),\\
&② \ [32, 39] \leftarrow A-: \ \ \mathcal{S}(2, 1, 3, p', 5, m = 00100),\\
&③ \ [40, 43] \leftarrow A-: \ \ \mathcal{S}(2, 1, 3, p', 6, m = 001010).
\end{aligned} \tag{16}$$

So far, we have discussed the case that τ is an odd number. If τ is even, its selecting process is exactly same as the odd case, except for the last select. Since the rightmost bit is 0 rather than 1, the last select in this case is using a select command to individually deal with the number τ. Its mask string is τ itself and the select command can be seen in (10).

Alg. 2 sketches LRQ for range detection within the scope of C1G2. Given the lower boundary τ, Line 1 gets the index set $\mathcal{R}(\tau) = \{r_i\}$ of 1s in binary representation, where $d(\tau)$ is the number of consecutive rightmost 1s. If the specific case that τ is equal to $2^{d(\tau)} - 1$ happens, a single select command with the mask string of $l' - d(\tau)$ zeros is able to choose all numbers in $[1, \tau]$, which is shown in Lines 2-4. Lines 5-13 deal with a more generalized case. For the first index r_1, Line 5 operates all numbers in $[1, 2^{r_1} - 1]$ by issuing only a single select command with the mask string of $l' - r_1$ zeros and the action AB. After that, except for the last $d(\tau)$ rightmost ones (if any), the reader repetitively checks each index r_i, and uses

Algorithm 2: Range Query LRQ(τ).

Input: The lower boundary τ.
Output: Is there any target tag: yes or no?

1 Get $\mathcal{R}(\tau) = \{r_i\}$ and $d(\tau)$;
2 **if** $\tau == 2^{d(\tau)} - 1$ **then**
3 | Flag $\leftarrow AB:\ \mathcal{S}(2, 0, 3, p', l' - d(\tau), 0)$;
4 **else**
5 | Flag$\leftarrow AB:\ \mathcal{S}(2, 0, 3, p', l' - r_1, 0)$;
6 | **for** $(i = 2;\ |\mathcal{R}(\tau)| - d(\tau);\ i++)$ **do**
7 | | Flag$\leftarrow A-:\ \mathcal{S}(2, 1, 3, p', l' - r_i, M(r_i) +' 0')$;
8 | **end**
9 | **if** $d(\tau) \geq 1$ **then**
10 | | Flag$\leftarrow A-:\ \mathcal{S}(2, 1, 3, p', l' - d(\tau), M(r_i))$;
11 | **else**
12 | | Flag$\leftarrow A-:\ \mathcal{S}(2, 1, 3, p', l', \tau)$;
13 | **end**
14 **end**
15 Query$-A:\ Q(0, 2, 0)$;
16 **if** *there is a reply* **then**
17 | return *yes;*
18 **else**
19 | return *no;*
20 **end**

a select to mask the numbers $[2^{r_{i-1}}, 2^{r_{i-1}} + 2^{r_i} - 1]$, where the mask string is the $l' - r_i - 1$ bits on the left of the r_i-th bit ($M(r_i)$) together with a bit '0'. Lines 9-13 depict the last select of LRQ. If τ is an odd number ($d(\tau) \geq 1$), the last $d(\tau)$ bits are ones and one mask can pick remaining numbers. If τ is an even number, only the number τ is left not-matching. Line 12 masks this single number with the value τ. To this end, the tags with the data being any one of $[1, \tau]$ are activated (flag is A) while others are silenced (flag is B). Line 15 issues a query command to inventory the active tags. If there is a reply, a target tag exists.

B. URQ & LURQ

So far, we have detailed the detection process of LRQ, which can easily expand to the other two queries: URQ and LURQ. For URQ (are there some data greater than an upper boundary), we just need to invert the actions of selects in Alg. 2. Namely, the actions AB and $A-$ are replaced with BA and $B-$, respectively. In this way, the tags with the data greater than the upper threshold are set to A, while others are set to B. After issuing the query command Query$-A: Q(0, 2, 0)$, only target tags will reply to the reader if any. For LURQ (are there some data in the interval $(\tau_L, \tau_U]$, where τ_L and τ_U are a lower boundary and an upper boundary, respectively), we only need to run the Alg. 2 twice with a small change. In the first run, we do LRQ(τ_U) as is according to Alg. 2 except for removing the query operation, which sets all tags with the data belonging to the interval $[1, \tau_U]$ to A, while others to B. In the second run, we do LRQ with the lower threshold τ_L, where all actions are set to $B-$. By this means, tags with the data in the interval $[1, \tau_L]$ are set to B; only the tags within the interval $(\tau_L, \tau_U]$ are A. The two runs are shown as follows:

$$\begin{aligned} &① \ [1, \tau_U] \leftarrow A : \text{Do } LRQ(\tau_U) \text{ Lines 1-14,} \\ &② \ [1, \tau_L] \leftarrow B : \text{Do } LRQ(\tau_L) \text{ with action } B-. \end{aligned} \tag{17}$$

C. Mask Combination

By investigating commodity RFID readers through their data sheets as well as 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 [18] 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 basic selects are compressed into a single one, saving the communication overhead by several times.

D. Performance Analysis

Now we discuss the execution time of LRQ. The other two (i.e., URQ and LURQ) can be easily derived based on LRQ. Given a lower boundary τ, the communication overhead of LRQ consists of two parts: 1) the selects issued by the reader for masking the numbers in the interval $[1, \tau]$ and 2) a query command together with an inventory frame for checking whether there is any tag reply. According to Alg. 2, the number of selects depends on $|\mathcal{R}(\tau)|$ and $d(\tau)$, where $|\mathcal{R}(\tau)|$ is the number of ones in τ's binary representation and $d(\tau)$ is the number of consecutive rightmost ones. Clearly, the number $f(\tau)$ of selects is:

$$f(\tau) = |\mathcal{R}(\tau)| - d(\tau) + 1. \tag{18}$$

This is a sharp decline in the number of selects. That is because $|\mathcal{R}(\tau)|$ is smaller than $\lceil \log_2 \tau \rceil$, which is much smaller than τ itself. For example, given $\tau = 5000$, the number $f(5000)$ of selects in LRQ is only 6, which is significantly smaller than 5000, compared with the basic range query. Besides, by taking the mask combination into account, the number of selects can be further reduced greatly. Hence, we have the execution time:

$$T(\tau) = \lceil \frac{f(\tau)}{\omega} \rceil \times t_s + t_q, \tag{19}$$

where ω is the number of masks that can be filled in one select, t_s is the time interval of issuing a select command by the reader, and t_q is the time interval of a query command together with an inventory frame for checking the presence of potential target tags. According to Eq. (19), the execution time of LRQ declines sharply. For example, consider 10,000 tags and $\tau = 5000$. LRQ spends only 2 selects ($w = 4$ when Alien readers [19] are adopted) together with an inventory to do the range detection, which is far superior to the overhead of 100 selects and 100 inventories (if each group has 100 tags on average), in comparison to SQ.

For URQ, its execution time is exactly same as that of LRQ. However, LURQ is slightly different. It needs two runs for selecting the numbers in the interval $(\tau_L, \tau_U]$, which are shown in Eq. (17). Hence, the execution time $T(\tau_L, \tau_U)$ of LURQ is:

$$T(\tau_L, \tau_U) = \lceil \frac{f(\tau_L) + f(\tau_U)}{\omega} \rceil \times t_s + t_q. \tag{20}$$

VI. Evaluation

A. Experimental Setup

We implement and evaluate our protocols with commodity RFID readers and tags. As shown in Fig. 1(a), four models of UHF RFID readers from two most experienced RFID suppliers are used in our experiments, including Impinj [18] and Alien [19]. Each reader is connected to a directional antenna that is with 8.5 dBic gain and operates at around 920 MHz. To better mimic a real RFID system and study the performance of the protocols in practice, we set up the experiments in a library, where up to 1000 commodity tags are used, which is shown in Fig. 1(b). We first investigate the capabilities of readers concerned by this study, in terms of the select and inventory operation. The experiment results show that these four readers support both of these operations. However, Impinj readers provide users with only character-level masking: the start position of the mask must be the $4i$-th bit (the first bit of each character). Instead, the Alien readers (e.g., ALN-F800, ALN-9900+) are able to do the bit-level mask, which is necessary for our protocols. Therefore, ALN-F800 is adopted to implement and evaluate our protocols in the experiments. We assert that the bit-level mask is specified by C1G2; the disability of Impinj readers 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 V-D, we have discussed the expected number of selects that is needed by RQ. Now we first study the number of selects in LRQ (URQ is the same as LRQ). As shown in Fig. 2(a), we vary the length of the threshold τ in binary representation from 1 to 16. Two cases are investigated: with mask combination (RQ-W) and without (RQ). Since the Alien readers [19] allow a select command to combine four masks, the factor ω in Eq. (19) is set to 4. As we can see, the (maximal and average) numbers of selects experience a near linear rise as the length of τ increases. This is intuitive as longer τ increases the probability of more binary ones. However, it does not mean that the large threshold needs more selects. It depends on the number of ones and the number of consecutive rightmost ones in the binary representation of τ. For example, when τ is equal to 20,000, LRQ needs 6 selects to do the mask. When τ is 2000, LRQ takes one more select, i.e., 7 selects, to do the mask. In addition, by taking the mask combination into account, this number will be further reduced (RQ-W). The positive results validate that LRQ and URQ are able to minimize the number of selects by combining the technologies of substring masking and mask combination.

In Fig. 2(b), we study the number of selects in LURQ, with respect to the lower boundary τ_L and the upper boundary τ_U. The lengths of these two boundaries range from 1 to 16, where $\tau_U > \tau_L$. Given a specific length pair of τ_L and τ_U, we plot the average number of selects that LURQ needs to cover the interval $(\tau_L, \tau_U]$. As we can see, the maximal number of selects is less than 17. If mask combination is considered, this number will drop to only 5. This small number well indicates that LURQ is time-efficient, regardless of the query interval.

(a) Readers. (b) Experimental setup.

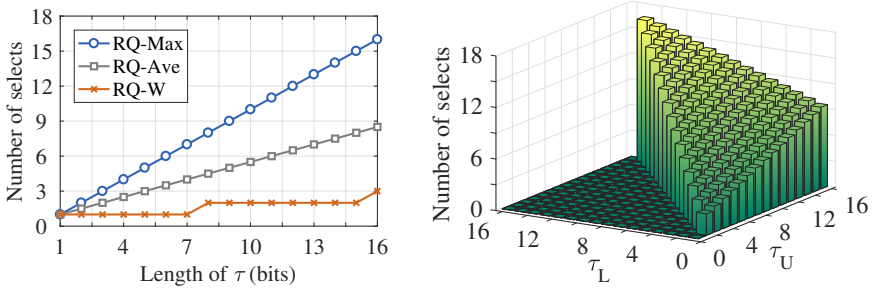
Fig. 1: Experimental Setup.

C. Time Efficiency

For range detection, high time efficiency is vital to give an early warning to users and help them reduce the potential risk. In this subsection, we study the execution time of our protocols. Since there is no prior work studying the problem of range detection within the scope of C1G2, we take exclusive collection (EC) as the baseline for comparison.

In Fig. 3, we compare the execution time of SQ and RQ (LRQ is adopted) with the baseline EC under three scenarios. For RQ, we set the threshold or the interval randomly in the domain of the data. In scenario 1, we set $n = 500$, $l' = 16$, $\rho = 0.1\%$, where n is the number of tags, l' is the length of the data in binary representation, and ρ is the ratio of the number of target tags to n. For SQ, we assume that each group has 20 tags on average. In scenario 2, we double the number n of tags, i.e., $n = 1000$, $l' = 16$, $\rho = 0.1\%$. Since one reader antenna cannot cover 1000 tags, we use two in the case of $n = 1000$. In scenario 3, we let the ratio ρ be zero, i.e., $n = 1000$, $l' = 16$, and $\rho = 0$, which means there is no target tag in the system. The execution time of the protocols are presented in Fig. 3. Take scenario 2 for example. The execution time of EC is 9.6s, which is the longest amongst the three protocols. SQ reduces the execution time to 3.2s since it silences most of tags with the same data. LRQ further reduces the execution time to 0.33s, which improves the time efficiency by near $30\times$, compared with EC. Similar conclusions can also be drawn in the other two scenarios: RQ performs the best, SQ follows, and EC is the worst.

Next we study in more details the impact of different parameters, including the number n of tags, the length l' of the data in binary representation, and the ratio ρ of the number of target tags to n. In Fig. 4(a), we show how n influences the execution time of EC, SQ, and RQ. We set l' to 16, ρ to 0.1%, and vary the number n of tags from 100 to 1000. When the number of tags is greater than 500, we use two antennas to do the range detection, as each antenna covers about 500 tags at most. Besides, assume that each group in SQ has 20 tags on average. For RQ, we adopt the worst case, four selects, which is shown in Section VI-B. As we can see, the execution time of EC increases with n. This is



(a) LRQ or URQ. (b) LURQ.

Fig. 2: The number of selects.

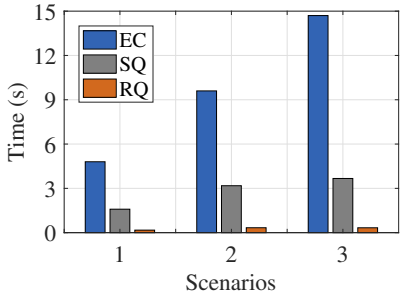
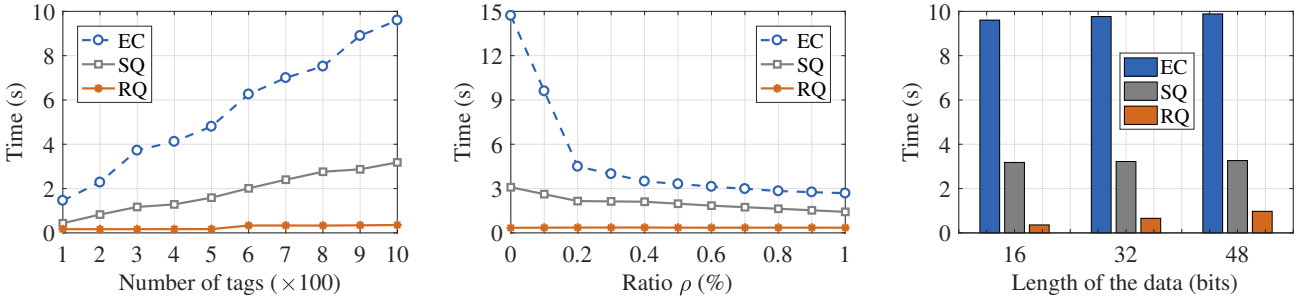


Fig. 3: Time efficiency.



(a) The number of tags. (b) The ratio of target tags to n. (c) The length of the data.

Fig. 4: The comparison of time efficiency.

intuitive as EC needs to individually communicate with each tag. The number of tags increases; the total communication time increases proportionally. Similar to EC, SQ also sees a rise trend over n. The difference is that the absolute value is much smaller. That is because it silences most tags with the same data. In contrast, the execution time of RQ remains stable, as it relies on the threshold τ. RQ improves the time efficiency by more than an order magnitude. For example, in the case of $n = 500$, EC takes 4.8s, SQ reduces the time to 1.6s, RQ further drops this time to 0.17s, producing a 28$\times$ performance gain.

In Fig. 4(b), we study the impact of the ratio ρ on the execution time of EC, SQ, and RQ. We fix n to 1000, l' to 16, and vary ρ from 0 (no target tag) to 1%, with a step of 0.1%. The results show that the execution time of EC and SQ both sees a downward trend as ρ increases. That is because in these two protocols once the reader detects a target tag, the execution process terminates. The large ρ increases the ratio of the number of target tags to n, as well as the probability that a target tag is detected. On the contrary, the execution time of RQ also remains stable since it aims to select all numbers within the given range, regardless of the ratio ρ.

In Fig. 4(c), we show the impact of l' on the execution time of EC, SQ, and RQ. We fix n to 1000, ρ to 0.1%, and vary l' from 16 to 48, with a step of 16 (as the reader collects the user data in word level). As we can see, the execution time of EC and SQ both experiences a very slight rise over l'. That is because when doing the inventory, EC and SQ need to do the read cycle and collect more data as l' gets longer. For RQ, the large l' increases the length of the boundary τ, as well as the number of selects. Even so, RQ is far superior to the baseline.

VII. Related Work

Range detection is to check if there are any tags that hold data with the value between an upper and lower boundary, which can be treated as a specific case of information collection. As one main branch of RFID research, information collection has attracted many attentions in recent years. The information can be the basic tag ID that exclusively labels each associated object, or the attributes of tagged goods for live query, or the sensing information in sensor-augmented RFID systems (e.g., the temperature of chilled food), etc. To achieve high efficiency, early research uses Aloha-based protocols [20]–[22] and tree-based protocols [8], [23]–[25] to identify tags in a time-efficient way. The basic idea of Aloha-based protocols is to let tags transmit data in different time slots. Each tag randomly selects a slot and only the slots chosen by exactly one tag can be used to collect ID information, which effectively avoids tag-to-tag collisions. The tree-based protocols apply a dynamic ID prefix of tag IDs to progressively split a tag set into smaller subsets until only one tag is left in each subset. This process is iteratively executed until all tags are successfully identified.

For other information collection, Chen et al. [11] propose an efficient multi-hash information collection protocol by using multiple hash functions to avoid tag collision during slot

assignment. Yue et al. [12] study the multi-reader RFID system and use a Bloom filter to determine which tags are covered by each reader. Qiao et al. [13] propose a polling-assisted protocols to collect the tag information from a subset of tags in an energy-efficient way. Liu et al. [15] propose an incremental polling protocol that sharply drops the polling vector from 96 bits to less than 2 bits, which greatly saves the polling overhead for information collection. In recent years, Liu et al. [14] investigate the problem of category information collection in a multi-category RFID system. Instead of repeatedly interrogating each tag, this work just samples each category by two steps: zooming into this category and isolating a tag at a small cost. Liu et al. [10] study the problem of range query that checks which interval the data held by each tag belong to. In other words, it aims to collect each tag's data in a coarse-grained way, which is with a different goal as our problem. Chen et al. [26] design a general framework, differential Bloom filter (DBF), to automatically detect anomalies in RFID systems. Unlike our work, DBF focuses on the anomaly event, such as missing tags, unknown tags, or cloned tags, rather than the specific data in each tag's memory. In spite of the advancement, these protocols cannot work in the commodity RFID systems due to C1G2 incompatibility. The reason is that these protocols make some ideal assumptions, e.g., hashing or building a filter, which however are not supported by C1G2.

VIII. Conclusion

This paper studies the problem of range detection in a commodity RFID system. By exploring the full potential of C1G2-compatible commands, we deliver two tailored solutions: selective query (SQ) and range query (RQ). SQ partitions the tag set into several groups and queries each group only once, which reduces the number of tag inventories to only the number of groups. RQ further reduces this number to exact one, by quickly separating target tags (if any) from the entire tag set with carefully designed selects. We implement these two protocols in a commodity RFID system, without any modification of hardware. Extensive experiments show that RQ is able to improve the time efficiency by an order of magnitude, in comparison to the baseline.

Acknowledgments

We would like to thank our shepherd, Prof. Chen Qian, and the anonymous reviewers for their valuable feedback on an earlier draft of this paper. This work is supported by National Natural Science Foundation of China (Nos. 61702257 and 61771236), Natural Science Foundation of Jiangsu Province (BK20170648), Jiangsu Key R&D Plan (Industry Foresight and Common Key Technology, BE2017154), Fundamental Research Funds for the Central Universities (14380066), and Collaborative Innovation Center of Novel Software Technology and Industrialization. Jia Liu and Lijun Chen are the corresponding authors.

References

[1] L. Shangguan and K. Jamieson, "The design and implementation of a mobile RFID tag sorting robot," in *Proc. of ACM MobiSys*, 2016, pp. 31–42.

[2] 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.

[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] H. Ding, J. Han, C. Qian, F. Xiao, G. Wang, N. Yang, W. Xi, and J. Xiao, "Trio: Utilizing tag interference for refined localization of passive RFID," in *Proc. of IEEE INFOCOM*, 2018, pp. 828–836.

[5] 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.

[6] 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*, vol. 6, no. 1, pp. 753–764, 2019.

[7] 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.

[8] J. Yu, W. Gong, J. Liu, and L. Chen, "Fast and reliable tag search in large-scale RFID systems: A probabilistic tree-based approach," in *Proc. of IEEE INFOCOM*, 2018, pp. 1133–1141.

[9] L. Xie, H. Han, Q. Li, J. Wu, and S. Lu, "Efficient protocols for collecting histograms in large-scale RFID systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 9, pp. 2421–2433, 2015.

[10] 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. 1124–1132.

[11] 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.

[12] 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.

[13] 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.

[14] J. Liu, S. Chen, B. Xiao, Y. Wang, and L. Chen, "Category information collection in RFID systems," in *Proc. of IEEE ICDCS*, 2017, pp. 2220–2225.

[15] J. Liu, B. Xiao, X. Liu, K. Bu, L. Chen, and C. Nie, "Efficient polling-based information collection in RFID systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 948–961, 2019.

[16] "EPC radio-frequency identity protocols generation-2 UHF RFID standard," GS1, ISO/IEC 18000-63, 2018.

[17] http://www.snopes.com/politics/business/rfid.asp.

[18] "Impinj Inc." http://www.impinj.com.

[19] "Alien Technology," http://www.alientechnology.com.

[20] 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.

[21] F. Schoute, "Dynamic frame length ALOHA," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 565–568, 1983.

[22] L. G. Roberts, "ALOHA packet system with and without slots and capture," *ACM SIGCOMM Computer Communication Review*, vol. 5, no. 2, pp. 28–42, 1975.

[23] M. Shahzad and A. X. Liu, "Probabilistic optimal tree hopping for RFID identification," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 293–304, 2013.

[24] N. Bhandari, A. Sahoo, and S. Iyer, "Intelligent query tree (iqt) protocol to improve RFID tag read efficiency," in *Proc. of IEEE ICIT)*, 2006, pp. 46–51.

[25] Y. Zheng and M. Li, "PET: Probabilistic estimating tree for large-scale RFID estimation," *IEEE Transactions on Mobile Computing*, vol. 11, no. 11, pp. 1763–1774, 2011.

[26] M. Chen, J. Liu, S. Chen, Y. Qiao, and Y. Zheng, "DBF: A general framework for anomaly detection in RFID systems," in *Proc. of IEEE INFOCOM*, 2017, pp. 1–9.