Multi-Range Query in Commodity RFID Systems

Yanyan Wang[†‡] Jia Liu[‡] Zhihao Qu[†] Shen-Huan Lyu[†‡§] Bin Tang[†] Baoliu Ye[‡]
[†]Key Laboratory of Water Big Data Technology of Ministry of Water Resources,
College of Computer Science and Software Engineering, Hohai University, Nanjing, China
[‡]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[§] Department of Computer Science, City University of Hong Kong, Hong Kong, China
{yanyan.wang, quzhihao, lvsh, cstb}@hhu.edu.cn, {jialiu, yebl}@nju.edu.cn

***Abstract*—Range Query (RQ) is to check whether there are any RFID tags with data beyond a given range. With about 46 billion RFID tags sold worldwide in 2023, time-efficient RQ becomes increasingly important for practical use, which can help users quickly pinpoint the target tags (if any) and give an early warning (e.g., fire alarm) to them for taking urgent actions and reducing the potential risk. However, existing work can deal with only a single range rather than multiple ranges that are very common in real-world applications. For example, foods in the refrigerator and the freezer have different temperature ranges for safe storing; treating them as one would probably give rise to query errors. In this paper, we study an under-investigated problem called *multi-range query*, which aims to achieve RQ in an RFID system with multiple query ranges. We propose a tailored protocol called anomalous tag identification (ATI) that quickly separates target tags from others and avoids querying all tags for saving communication overhead. In ATI, we design a fixed-length encoding vector together with standards-compliant select commands to deal with different ranges individually, without the need for any hardware modification. We implement the proposed protocols in commodity RFID systems. Experimental results show that ATI is superior to the baseline under different parameters, in terms of the time efficiency and space efficiency.**

***Index Terms*—RFID, multi-range query, C1G2, time efficiency**

I. Introduction

Radio frequency identification (RFID) has been widely used in a variety of applications, including library inventory [1], [2], human-machine interaction [3]–[7], object localization and tracking [8]–[12], warehouse management [13]–[19], etc. In these applications, each object is attached with an RFID tag to provide unique identification and additional information for item-level intelligence. To meet the increasing demand for real-time data collection and digital business, RFID sensors have been developed and integrated into RFID tags. In a sensor-augmented RFID system, the tag is capable of carrying some dynamic sensing data, such as temperature and humidity. By communicating with tags and collecting their data, we can achieve real-time information about the state of tagged objects or monitor their surroundings.

Range Query (RQ) is to check whether there are any RFID tags with data beyond a given range, which is one of research branches of RFID and plays a major role in RFID-enabled applications. In 2023, with about 46 billion RFID tags sold worldwide, time-efficient RQ becomes increasingly important for practical use, which can help users quickly pinpoint the target tags (if any) and give an early warning (e.g., fire alarm) to them for taking urgent actions and reducing the potential risk. For example, 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) will be activated to protect people and assets.

In recent years, some advanced work has been proposed to address the range query problem in a time-efficient way [20], [21]. However, existing range query work can deal with only a single range rather than multiple ranges that are very common in real-world applications. For example, foods in the refrigerator and the freezer have different temperature ranges for safe storing — frozen food typically falls within the range from $-24°C$ to $-18°C$, while refrigerated food requires temperatures between $2°C$ and $8°C$. If we treat them as one, the system probably give rise to query errors. For example, if we focus on only the interval $[-24°C, -18°C]$ of frozen food, the refrigerated food with the normal temperature range $[2°C, 8°C]$ will respond to the reader, leading to extra communication overhead and even incorrect queries.

An intuitive solution for multi-range anomalous tag identification is to collect data from all tags and check if any tags have data beyond their expected range. This works but suffers from long time delays in large systems with numerous tags, especially when the number of anomalous tags is far less than that of all tags. In recent years, some advanced approaches have been proposed to expedite the information collection process [22]–[25]. These solutions, however, have two limitations: they cannot be implemented in a commodity RFID system since they are incompatible with the EPCglobal Class1 Gen2 (C1G2) [26]; they need to query all tags, which is time-consuming.

In this paper, we study the under-investigated problem of *multi-range query*, which aims to achieve RQ in an RFID system with multiple query ranges. We propose a tailored protocol called anomalous tag identification (ATI). The basic design idea is to partition the entire tag set into multiple groups, each of which contains tags with data falling into the

*Corresponding authors: Zhihao Qu and Shen-Huan Lyu.

979-8-3315-4940-4/25/$31.00 © 2025 IEEE

same range. When monitoring a particular range, we separate the anomalous tags (with data beyond the specific range) from other tags within the same group, which enables us to avoid the interference of tags in other groups and identify only the anomalous tags (if any) of this group. To achieve this goal, we design a fixed-length encoding vector together with standards-compliant select commands to deal with each range (group) in turn – the encoding vector helps save the number of masks (i.e., the communication overhead) and the select commands are used to pick tags that match a given mask. By this means, we can silence a majority of non-target tags and identify anomalous tags within a group by carrying out an inventory frame with the standards-compliant query command. We implement a prototype in commercial RFID systems with 500 tags and experimental results show the good performance of ATI. The main contributions of this paper are three-fold.

- To the best of our knowledge, we are the first to study multi-range query, which helps users quickly and accurately identify anomalous tags and reduce the potential losses in an RFID system with multiple query ranges.
- We propose a protocol called anomalous tag identification (ATI) that quickly separates target tags from others and avoids querying all tags for saving communication overhead by designing a fixed-length encoding vector together with standards-compliant select commands.
- We implement a prototype of ATI in a commodity RFID system with 500 tags. Extensive experiments show that ATI improves the time efficiency of range identification by 1.3 $\times$, compared with the baseline, with 10 ranges, 300 tags, and 5 anomalous tags in one anomalous range.

The rest of the paper is organized as follows. Section II formulates the problem of multi-range query. Section III presents ATI. Section IV evaluates the performance of ATI. Section V discusses the related work. Finally, Section VI concludes this paper.

II. PROBLEM FORMULATION

A. Problem Definition

We consider a sensor-augmented system that consists of a reader and a number of tags. Each tag has a unique ID that indicates the object it attaches to. By collecting relevant information from tags, the reader can get the real-time status of the surrounding environment (e.g., temperature, humidity, etc.). Different objects may have distinct requirements, leading to varying data ranges of interest. Consequently, tags attached to different objects may store data within different ranges. *The problem of multi-range query is to identify tags with data beyond their expected ranges.* These tags are referred to as anomalous tags (or target tags). Specifically, consider a tag set $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$, where each subset $\Gamma_i = \{t_1, t_2, \ldots, t_{n_i}\}$ $(1 \leq i \leq m)$ represents a distinct group of tags and m is the number of given normal data ranges. For Γ_i, the expected data range of each tag t_j that holds data d_j, $1 \leq j \leq n_i$, is $D_i = [l_i, u_i]$, where l_i and u_i are the lower and upper boundaries of D_i, respectively. We assume that the value of d_j is a non-negative integer, each of which represents a dynamic status of surroundings. For instance, the general temperature range for frozen food, which spans from $[-24^\circ C, -18^\circ C]$ with a resolution of $0.01^\circ C$, can be mapped to the integer interval [0, 600]. Our objective is to identify anomalous tags within Γ_i $(1 \leq i \leq m)$, which refers to tags whose data d_j with data beyond its expected range D_i. High time efficiency and accuracy are two essential requirements that enable accurate and timely warnings to users, minimizing potential losses.

B. Basic Solutions

To identify anomalous tags in Γ_i $(1 \leq i \leq m)$, an intuitive solution is to query each tag in the system and check whether its data falls within its corresponding expected range. If yes, the tag is considered normal; otherwise, it is anomalous. However, this basic collection results in high time overhead as it requires inventorying the entire tag set for each identification, even when monitoring anomalies within a limited number of ranges of interest rather than all ranges.

The recent *Range query* (RQ) [20], [21] can pinpoint anomalous tags in Γ_i when m is 1, but it struggles in a multi-range scenario, because it cannot distinguish anomalous tags of one range from normal tags of other ranges, leading to false alarms. To make RQ work properly in our problem, we make some modifications on this work and give a baseline protocol for comparison in the following.

Given a data range $D_i = [l_i, u_i]$, RQ first partitions Γ into two subsets: $\Gamma_i^{'}$, which contains all tags whose data falls within D_i, and $\Gamma - \Gamma_i^{'}$, which contains the rest of tags. Note that $\Gamma_i^{'} \nsubseteq \Gamma_i$ as it is possible for certain anomalous tags from other ranges to also fall within D_i. Instead of querying tags in $\Gamma - \Gamma_i^{'}$ that includes the anomalous tags (if any) of Γ_i and tags from $\Gamma - \Gamma_i$, the modified RQ approach (mRQ) directly queries the tags in $\Gamma_i^{'}$ and compares them with the expected range Γ_i. If $\Gamma_i \subseteq \Gamma_i^{'}$, all tags in Γ_i are considered normal; otherwise, any anomalous tags within $\Gamma_i - \Gamma_i^{'}$ can be identified. The partition process is accomplished by using the `Select` command specified in EPCglobal Class1 Gen2 (C1G2) [26]. This command is issued before the inventory operation, enabling a reader to selectively choose a subset of tags that participate in the subsequent query, optimizing the query process by narrowing down the scope of tags involved.

Although mRQ can successfully identify anomalous tags in Γ_i, the querying process for tags in $\Gamma_i^{'}$ leads to time overhead. In practical scenarios, the number of anomalous tags in Γ_i is typically small, and $\Gamma_i^{'}$ closely approximates Γ_i. In addition, when all tags in a system are associated with multiple ranges, and all ranges are of interest, mRQ incurs an even higher time overhead than the basic collection.

III. ANOMALOUS TAG IDENTIFICATION

In this section, we propose an anomalous tag identification protocol (ATI), which builds a fixed-length encoding vector incorporates grouping information to effectively query only the anomalous tags. This improves the time efficiency of identifying anomalous tags in scenarios where the normal data

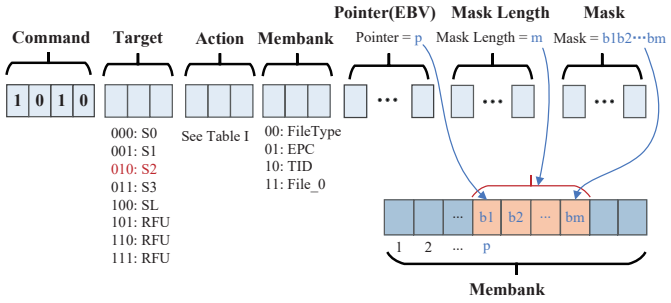

Fig. 1. Select Command

of tags spans multiple ranges. In what follows, we first discuss the functionalities within the scope of the C1G2 standard and then describe the details of ATI.

A. C1G2 Function

As specified in C1G2 [26], a reader can perform a select operation to choose a subset of tags for querying. We now describe how to use the `Select` command to select a subset of tags as required and subsequently execute the inventory operation using the `Query` command.

1) Select Command: One `Select` command actually sets the inventoried flags of tags to two distinct states by specifying a mask string. By issuing one or more `Select` commands, we have the flexibility to choose a subset of tags as required. The command consists of six fields, as shown in Fig. 1.

• `Membank`(b), `Pointer`(p), `Length`(l), `Mask`(m). These four fields collectively determine the matching of tags. `MemBank` specifies the memory bank for comparison, with four distinct memory banks available. `Membank`-0 is usually reserved, `Membank`-1 stores the Electronic Product Code (EPC), `Membank`-2 stores TID (tag- and manufacturer-specific data), and `Membank`-3 is user memory for customized data. In this work, we have fixed the `MemBank` to 3, as it is a common choice for many applications. `Pointer` indicates the starting bit position in the chosen memory bank. `Length` determines the length of `Mask`, which is a customized bit string based on application requirements. Upon receiving a `Select` command, a tag compares the data in its designated `Membank` starting from the `Pointer` position to the subsequent `Length` bits. If the data matches the specified `Mask` sub-string, the tag is deemed a match; otherwise, it is considered a non-match.

• `Target`(t), `Action`(a). These two fields determine the modification of flags for a group of tags. By applying masks and modifying the flags, the `Select` command sets the flags of matching tags to a specific value, while the flags of non-matching tags are set to the opposite value. `Target` represents either the selected flag (SL) or the inventoried flag, which serves as an access control indicator for the tags. The first bit in `Target` indicates which flag will be modified (0 for inventoried flag and 1 for SL). The remaining two bits indicate the session in which the tags will participate. C1G2 defines four sessions, each with a different persistent time that the tag holds the inventoried flag. These two bits are irrelevant for the selected flag (SL). In this paper, we utilize the inventoried flag in session 2 as a metric to demonstrate our protocol design. `Action` determines how the chosen flags will be set or modified. Table I lists eight actions, allowing matching and non-matching tags to assert or deassert their SL flags, or set their inventoried flags to A or B.

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

2) Query Command: The `Query` command initiates and specifies an inventory round. After the `Select` commands have set the special flags of the tags, the subsequent `Query` command uses these flags to determine which tags participate in the inventory round. In this work, we concern three fields: `Sel`, `Session`, and `Target`. The `Sel` field consists of two bits that determine which tags will respond to the reader: 00_2 and 01_2 indicate all matching tags selected by the previous `Select` command, 10_2 indicates tags with a deasserted (∼SL) flag, and 11_2 indicates tags with an asserted SL flag. Since SL is not used in this paper, this field is always set to 0. `Session` selects the session for the inventory frame. In this paper, we use session 2 (S2), and accordingly, the value is set to 2 (10_2). `Target` determines the participation of tags in the inventory frame: 0 for flag A and 1 for flag B. After being successfully queried, tags will have their inventoried flags inverted, switching between A and B.

B. ATI Protocol

ATI determines if any tags, whose data should normally fall within the range $D_i = [l_i, u_i]$, are now beyond that range. To achieve this, ATI excludes tags belonging to $\bigcup_{j=1,j\neq i}^{m-1} \Gamma_j \cup \Gamma_i^{'}$ and queries the remaining tags. $\Gamma_i^{'}$ contains all tags whose data falls within D_i. Next, we describe the process of selecting tags in different scenarios: data between the upper and lower boundaries of D_i (LUS), data not greater than u_i (LS), and data not less than l_i (US). We first detail LS and then generalize to US and LUS. Moreover, to suppress interference tags from $\bigcup_{j=1,j\neq i}^{m-1} \Gamma_j \cup \Gamma_i^{'}$, ATI employs a fixed-length encoding vector to strike a balance between time and space efficiency.

1) Encoding Vector: For clarity, we will use τ as a unified symbol to represent the upper boundary u_i. In range query works, the number of selects required to label tags whose data falls within the range $[0, \tau]$ depends on the value of τ. One observation is that τ often corresponds to a decimal value of many trailing zeros. For example, consider a temperature range for a certain type of food, denoted as $[a°C, b°C]$, with a resolution of $0.01°C$. This range can be mapped to the integer interval $[0, c00]$, where $c = b - a$. If the resolution is $0.001°C$, the interval will be $[0, c000]$. In common cases, the difference value c between b and a is relatively small. To optimize the

number of selects and reduce memory space requirements, we design a fixed-length encoding vector that takes advantage of the trailing zeros in the integer representation of the range.

We use a bit vector set $\mathcal{V} = \{v_0, v_1, \cdots, v_9\}$ to represent the digits from 0 to 9. Each digit is encoded using a 4-bit vector. The encoding scheme follows the principle of selecting 9 binary strings that have a minimal number of 1s or consecutive 1s, ranging from 0 to $2^4 - 1$. This is based on the fact that, according to [20], the number of selects for a binary representation of τ is $f(\tau) = |\mathcal{R}(\tau)| - d(\tau) + 1$, where $|\mathcal{R}(\tau)|$ is the number of ones in τ's binary representation and $d(\tau)$ is the number of consecutive rightmost ones. The specific encoding for the digits 0 to 9 is as follows:

$$0: \text{ '0000'}, 1: \text{ '0001'}, 2: \text{ '0010'}, 3: \text{ '0011'}, 4: \text{ '0100'},$$
$$5: \text{ '0111'}, 6: \text{ '1000'}, 7: \text{ '1001'}, 8: \text{ '1010'}, 9: \text{ '1111'}.$$

Hence, we have $\mathcal{V} = \{$'0000', '0001', '0010', '0011', '0100', '0111', '1000', '1001', '1010', '1111'$\}$, and we use $EV(d)$ to represent the encoding vector of data d.

The encoding vector's design directly affects the number of select commands, which consequently influences the overall execution time. To evaluate its performance, we conduct a simulation and compare ATI with existing range query protocols (RQ, EnRQ4, EnRQ10, and EnRQ16) for selecting all numbers in the range $[0, \tau]$. The results are shown in Fig. 2. We can see that ATI requires fewer selects than RQ, EnRQ4, and EnRQ16. Although EnRQ10 requires fewer selects than ATI, it requires more memory space. For example, when τ is 20,000, the length of the encoding vector in ATI is $5 \times 4 = 20$ bits, while EnRQ10 requires $5 \times 9 = 45$ bits, which is more than twice the length of the encoding vector. Despite this, the number of selects required by EnRQ10 is only one less than that of the encoding vector in some cases. This indicates that the encoding vector in ATI offers a good trade-off between time efficiency and memory space.

Recall that when selecting tags with data in one range, it is necessary to silence the tags associated with other ranges. To accomplish this, we assign a unique group string to each group of tags associated with the same range. This group string remains fixed, and selecting tags from a range only requires a single select command. Therefore, to represent the m ranges, we use the space-saving binary strings from 1 to m. For example, if $m = 3$, the group vector, denoted by $\{G()\}$, is $\{01_2, 10_2, 11_2\}$. We combine the encoding vector for data information with the group vector for grouping information. For a tag associated with the first data range (belonging to the first group), with a collected data 45, the final encoding result is '0101000111'.

2) Design of Mask: In this subsection, we will discuss the masking process with the data set of $\{d_i\}$, which is represented as $\{EV(d_i)\}$ in a tag's memory. We denote the number of digits in τ as l_τ, and the data of interest ranges from 0 to $10^{l_\tau} - 1$. Consider three special cases that $\tau = \{10, 6, 4\} \times 10^{w-1} - 1$, where $w < l_\tau$. The $EV(\tau)$ is:

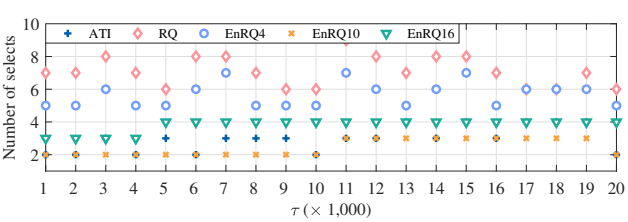

Fig. 2. Number of selects.

$$\overbrace{0\,0\,0\,0}^{4}\ldots\overbrace{0\,0\,0\,0}^{4}\ \overbrace{0\ x\ 1\ 1}^{4}\ldots\overbrace{1\,1\,1\,1}^{4}. \tag{1}$$
$$\underbrace{\qquad\qquad}_{(l_\tau - w)\times 4} \quad \underbrace{\qquad\qquad}_{w\times 4}$$

Clearly, the left most $(l_\tau - w) \times 4 + 1$ bits are all zeros. When $x = 1$, the rightmost $4w - 1$ bits are all ones. For any number larger than $6 \times 10^w - 1$, the rightmost $4w - 1$ digits start over, and next digit is incremented. As a result, the leftmost $(l_\tau - w) \times 4 + 1$ bits will not all be zeros. Conversely, for numbers smaller than $6 \times 10^w - 1$, the leftmost $(l_\tau - w) \times 4$ bits must be zeros, as any non-zero value in those bits would make the number larger than $6 \times 10^w - 1$. Therefore, we can use the leftmost $(l_\tau - w) \times 4 + 1$ bits as a mask to select all numbers in the range of $[1, \tau]$, effectively activating the tags associated with data within that range. Similarly, when $x = 0$, we can use the leftmost $(l_\tau - w) \times 4 + 2$ bits as a mask.

For the special cases mentioned above, we can make some modifications to generalize them and cover all numbers in the range $[1, \tau]$. Given any τ, it can be expressed as:

$$0\ldots0\ X_1\ldots X_2\ldots\underbrace{0\ldots0}_{z(\tau)}\ \underbrace{Y\ldots9}_{d(\tau)}, \tag{2}$$

where $0 < X_1, X_2 < 9$, and Y is 3 or 5. $d(\tau)$ counts consecutive rightmost digits in τ that are 9 or a single 3 (or 5) followed by some 9s, and $z(\tau)$ counts consecutive digit 0s before the consecutive rightmost 9s. Correspondingly, $EV(\tau)$ can be expressed as:

$$\overbrace{0\,0\,0\,0}^{4}\overbrace{b_1\ b_2\ b_3\ b_4}^{4}\ldots\overbrace{0\,0\,0\,0}^{4}\ldots\overbrace{0\,0\,0\,0}^{4}\ \overbrace{0\ x\ 1\ 1}^{4}\ldots\overbrace{1\,1\,1\,1}^{4}, \tag{3}$$
$$\uparrow_1\ \uparrow_2\ \uparrow_3\ \uparrow_4 \qquad \underbrace{\qquad\qquad}_{z(\tau)} \qquad \underbrace{\qquad\qquad}_{d(\tau)}$$

where b_{1-4} can be 0 or 1.

For LS, the main idea is to find a mask that covers the largest sub-interval of $[0, \tau]$ to minimize the number of selects required. In our design, the upper boundary value for this interval is given by $(u_1+1) \times 10^{l_\tau - 1} - 1$, where u_1 depends on the first non-zero digit in $EV(\tau)$. If the first non-zero bit is at position b_2, we set this bit to 0, resulting in $b_1b_2b_3b_4 =$ '$00xx$', where x represents a binary bit. Then, u_1 is determined as the value in $\mathcal{V}$ where the first two bits are 0. In this case, '0000', '0001', '0010', and '0011' correspond to $u_1 = 0$, $u_1 = 1$, $u_1 = 2$, and $u_1 = 3$, respectively. We denote the smallest and largest value as u_1 and u_1', respectively. For example, given a number 8053. For the first digit 8, the encoding vector $v_8 =$ '1010'. By setting the first bit of v_8 to 0, the largest value in $\mathcal{V}$ with the first one bit as 0 is '0111', which corresponds to

$u'_1 = 5$. Hence, the upper boundary value of the sub-interval can be calculated as $(u'_1 + 1) \times 10^{l_\tau - 1} - 1 = 5999$. The mask string is the left bits before the first '1' in $EV(\tau)$, together with another '0'. In this example, the mask string is '0', which allows selecting all the values within the range [0, 5999].

After that, we can repeat this process recursively. In the i-th select ($i > 1$, $k = 1$), k represents the position of the digits in τ, starting from the most significant digit with position 1. We mask the numbers in the sub-interval:

$$\left[u_i \times 10^{l_\tau - 1}, (u'_i + 1) \times 10^{l_\tau - 1} - 1\right], \tag{4}$$

where u_i and u'_i are the smallest and largest values in $\mathcal{V}$, that should match the string obtained from the digits of $v(k)$ before the i-th bit, with an additional 0 added at the i-th position. For $k > 1$, S_{k-1} represents the number formed by the digits before the k position of τ, calculated as $\sum_{j=1}^{k-1} X_j \times 10^{l_\tau - j}$. We can mask the numbers within this sub-interval:

$$\left[\sum_{j=1}^{k-1} X_j 10^{l_\tau - j} + u_i 10^{l_\tau - 1}, \sum_{j=1}^{k-1} X_j 10^{l_\tau - j} + (u'_i + 1) 10^{l_\tau - 1}\right], \tag{5}$$

In the second selection of 8053, setting the second '1' of v_8 to 0 yields $u_2 = 6$ and $u'_2 = 7$. We can use the mask string '100' to select the numbers in $[6000, 7999]$. For each digit, we process each bit that is a '1' starting from the first '1' until $u'_i = X_i - 1$. In this case, $u'_2 = 7$. Moving on to the other digits in $EV(\tau)$, we skip the leading zeros before the first '1' of the other digits in $EV(\tau)$, which is the first '1' of v_5, resulting in $u_3 = 0$ and $u'_3 = 3$. At this point, $k = 3$, and $S_{k-1} = 8000$. The mask string '1010000000' (bits of $EV(\tau)$ before the first '1' of v_5 with a '0' added) to select the numbers within the range $[8000, 8039]$.

When $z(\tau) > 0$ and $d(\tau) > 0$, we can skip the $4z(\tau)$ bits in $EV(\tau)$ without needing extra selects, and deal with the rightmost consecutive 1s in $EV(\tau)$ by one select. For example, given the number 8059, three mask strings are used in total:

$$\begin{aligned} &① \ [\quad 0, 5999] \leftarrow mask : 0, \\ &② \ [6000, 7999] \leftarrow mask : 100, \\ &③ \ [8000, 8059] \leftarrow mask : 101000000. \end{aligned} \tag{6}$$

If $z(\tau) > 0$ and $d(\tau) = 0$, a new string is required to mask the number τ. Given the number 460, three masks are required:

$$\begin{aligned} &① \ [\quad 0, 399] \leftarrow mask : 00, \\ &② \ [400, 459] \leftarrow mask : 01000, \\ &③ \ [460] \leftarrow mask : 010010000000. \end{aligned} \tag{7}$$

For clarity, we provide the corresponding mask substrings to be added for each digit in Table II. The submask differs when $k < l_\tau$ and $k = l_\tau$. Considering the number obtained by reversing the digits of 460, which is 064, the masks are:

$$\begin{aligned} &① \ [\ 0, 59] \leftarrow mask : 0, \\ &② \ [60, 63] \leftarrow mask : 100000, \\ &③ \ [64] \leftarrow mask : 1000010. \end{aligned} \tag{8}$$

Table II: Submasks of Each Digit

Digits	Submasks ($k < l_\tau$)	Submasks ($k = l_\tau$)
0	-	{0000}
1	{0000}	{000}
2	{000}	{000, 0010}
3	{000, 0010}	{00}
4	{00}	{00, 010}
5	{00, 0100}	{0}
6	{0}	{0, 1000}
7	{0, 1000}	{0, 100}
8	{0, 100}	{0, 100}
9	{0, 10}	-

Algorithm 1: Get Masks in ATI

Input : The boundary τ
Output: The mask set MS to select numbers in $[0, \tau]$

1 Get $EV(\tau)$, l_τ, $d'(\tau)$;
2 **if** $\tau = \{10, 6, 4\} \times 10^{w-1} - 1$ **then**
3 add $\underbrace{'0 \ldots 0'}$ to MS;
4 **else**
5 initialize MS as an empty set;
6 **for** $i = 1$ **to** $l_\tau - d'(\tau) - 1$ **do**
7 **if** *digit at position i is not 0* **then**
8 add Premask(i) + Submask(i) to MS;
9 **end**
10 **end**
11 **if** $d'(\tau) > 0$ **then**
12 add Premask($l_\tau - d'(\tau)$) + Submask($l_\tau - d'(\tau)$) to MS;
13 **end**
14 **else**
15 add Premask($l_\tau - d'(\tau)$) + SubmaskLast($l_\tau - d'(\tau)$) to MS;
16 **end**
17 **end**
18 **return** MS;

Alg. 1 outlines the masking process to generate a mask set MS for selecting numbers in the interval $[1, \tau]$. Given the boundary τ, we first get $EV(\tau)$. Then, we get the length l_τ of $EV(\tau)$ and the number of consecutive 9 in $EV(\tau)$, denoted as $d'(\tau)$. If τ satisfies the condition $\tau = \{10, 6, 4\} \times 10^{w-1} - 1$, the algorithm adds a single mask string consisting of zeros before the first 1 in $EV(\tau)$ (Lines 2-3). This mask string is sufficient to select all numbers in the interval $[1, \tau]$. For the general case, outlined in Lines 5-18, the algorithm iterates through the digits of τ. It skips digit 0 for the first $l_\tau - d'(\tau) - 1$ digits and constructs a mask string by combining the Premask, which consists of the $4(i-1)$ bits of $EV(\tau)$, with the Submask of each digit. For the $(l_\tau - d'(\tau))$th digit, it adds the Premask string, which consists of the $4(l_\tau - d'(\tau) - 1)$ bits of $EV(\tau)$, and the Submask string of this digit as when $k = l_\tau$. Finally, the algorithm returns the resulting mask set MS.

3) ATI Description: To identify anomalous tags associated with m ranges, ATI follows a process that involves m rounds, each comprising several selects and a query. We first describe the process for LS and then generalize it to US and LUS. To silence tags with data in the range $[0, \tau]$, we use carefully designed `Select` commands to set their inventoried flags.

The `Select` command can be simply denoted as:

$$\mathcal{S}(\underbrace{t}_{\text{Target}}, \overbrace{a}^{\text{Action}}, \underbrace{b}_{\text{MemBank}}, \overbrace{p}^{\text{Pointer}}, \underbrace{l}_{\text{Length}}, \overbrace{m}^{\text{Mask}}). \tag{9}$$

The six fields and the masking process for numbers in $[0, \tau]$ have been explained in Section III-A and Section III-B2, respectively. Using the mask set obtained from Alg.1, we design the corresponding `Select` commands to set the inventoried flags of matching tags to B and the flags of other tags to A. For the first mask, we silence the matching tags and activate the rest. The `Select` is:

$$\text{Flag} \leftarrow BA : \mathcal{S}(2, a{=}4, 3, p', len_1, mask_1), \tag{10}$$

where $a = 4$ denotes the action of BA, $mask$ is the mask set obtained from Alg. 1, len_i is the length of the i-th mask string, and $mask_i$ is the i-th mask string. Tags matching the substring specified by $mask_1$ from the p'-th bit will have their inventoried flag set to B, while the remaining tags will have their flags set to A. For the rest masks in masks set, we only need to silence the matching tags. The `Select` is:

$$\text{Flag} \leftarrow B- : \mathcal{S}(2, a{=}5, 3, p', len_i, mask_i), \tag{11}$$

where $a = 5$ means the action of $B-$.

So far, we have discussed silencing tags whose data fall within the range $[0, \tau]$. To silence tags with data in $[\tau_L, \tau_U]$ (LUS), we first silence tags with data in the range $[0, \tau_U]$ (LS). Then, we use $A-$ instead of $B-$ to silence the tags with data larger than τ_L (US). This ensures that only the tags within the interval $[\tau_L, \tau_U]$ with the flag set to B.

Alg. 2 shows how ATI identifies anomalous tags within the range $[\tau_L, \tau_U]$. The mask sets $maskU$ and $maskL$ are obtained using the *getMasks* function in Alg. 1 (Lines 1 and 2). ATI selects tags with data in the range $[0, \tau_U]$ using $B-$ action, except for the first command which uses the BA action (Line 5). Subsequently, ATI selects tags with data in the range $[0, \tau_L]$ using $A-$ commands (Line 8). Tags within $[\tau_L, \tau_U]$ are set to B, while tags with data beyond this range are set to A. Further, ATI silences tags with data beyond the range $[\tau_L, \tau_U]$ using $-B$ action based on $maskG$, the mask string of the group vector (Line 10). Finally, ATI performs a query command to inventory the tags with flag A, and adds any replied tags to the anomalous tag set $\mathcal{A}$ (Line 12).

Algorithm 2: ATI of Each Range

Input: The boundaries τ_U and τ_L.
Output: The anomalous tag set $\mathcal{A}$.

1 $maskU$ = $getMasks(\tau_U)$;
2 $maskL$ = $getMasks(\tau_L)$;
3 Flag $\leftarrow BA : \mathcal{S}(2, 4, 3, p', len_{U_1}, maskU_1)$;
4 **for** $(i = 2;\ sizeof(maskU);\ i++)$ **do**
5 | Flag $\leftarrow B- : \mathcal{S}(2, 5, 3, p', len_{U_i}, maskU_i)$;
6 **end**
7 **for** $(i = 1;\ sizeof(maskL);\ i++)$ **do**
8 | Flag $\leftarrow A- : \mathcal{S}(2, 1, 3, p', len_{L_i}, maskL_i)$;
9 **end**
10 Flag $\leftarrow -B : \mathcal{S}(2, 2, 3, p_G, len_G, maskG)$;
11 Query $\leftarrow A : Q(0, 2, 0)$;
12 **if** *there is a reply* **then**
13 | Add tags to $\mathcal{A}$;
14 | return $\mathcal{A}$;
15 **else**
16 | return *no*;
17 **end**

C. Performance Analysis

Now, we discuss the execution time of ATI. According to Alg. 2, the time for identifying anomalous tags of each range consists of three parts: select time, query time, and collect time. The select time is the duration required for issuing select commands, the query time is the time taken to issue a query command together with an inventory frame, and the collect time refers to the time to collect the IDs of the tags that replied to the query command. Based on Alg. 1, the number of selects for τ, denoted as $f'(\tau)$, depends on $d'(\tau)$ and the digits of τ.

$$f'(\tau) = \sum_{i=1}^{l_\tau - d'(\tau)} s_d(i), \tag{12}$$

where $s_d(i)$ is the number of selects of the ith digit of τ, which is shown in Tab. II. Consequently, the execution time for processing a range $D_i = [l_i, u_i]$ can be calculated as:

$$T(D_i) = (f'(l_i) + f'(u_i) + 1) \times t_s + t_q + n_{a_i} \times t_{id}, \tag{13}$$

where t_s is the time interval of issuing a select command by the reader, t_q is the time interval of a query command together with an inventory frame, t_{id} signifies the time to collect the IDs of the tags that responded to the query command (i.e., the abnormal tags), and n_{a_i} indicates the number of abnormal tags in the i-th group. The total execution time for identifying all abnormal tags in m ranges can be calculated as:

$$T = \sum_{j=1}^{m} \left\{ \Big(\sum_{i=1}^{l_{u_i} - d'(u_i)} s_d(i) + \sum_{i=1}^{l_{l_i} - d'(l_i)} s_d(i) + 1 \Big) \times t_s + t_q + n_{a_i} \times t_{id} \right\}. \tag{14}$$

D. Identification after Detection

ATI returns IDs for anomalous tags, facilitating anomalous object identification. However, collecting IDs is time-consuming when there are many anomalous tags of one range, leading to delays in detecting anomalies in other ranges. To address this, we can first detect anomalies in the ranges of interest and then identify the anomalous tags. During detection, the reader collects the ID of just one tag after issuing the

Algorithm 3: Enhanced ATI of Each Range

Input: The boundaries τ_U and τ_L.
Output: The anomalous tag set $\mathcal{A}$.

1 $maskU$ = $getMasks(\tau_U)$;
2 $maskL$ = $getMasks(\tau_L)$;
3 Flag $\leftarrow BA$: $\mathcal{S}(2, 4, 3, p', len_{U_1} + len_G, maskG + maskU_1)$;
4 **for** $(i = 2;\ sizeof(maskU);\ i++)$ **do**
5 | Flag $\leftarrow B-$: $\mathcal{S}(2, 5, 3, p', len_{U_i} + len_G, maskG + maskU_i)$;
6 **end**
7 **for** $(i = 1;\ sizeof(maskL);\ i++)$ **do**
8 | Flag $\leftarrow A-$: $\mathcal{S}(2, 1, 3, p', len_{L_i} + len_G, maskG + maskL_i)$;
9 **end**
10 Query $\leftarrow B : Q(0, 2, 1)$;
11 **if** *there is a reply* **then**
12 | Add tags to $\mathcal{A}$;
13 | return $\mathcal{A}$;
14 **else**
15 | return *no*;
16 **end**

query. If a reply is received, it indicates anomalies in the range; otherwise, the range is considered normal. Subsequently, the reader focuses on collecting IDs of anomalous tags associated with the identified anomalous ranges. Consider an anomalous range. In some extreme scenarios, where most of the tags in an anomalous range are affected by the surrounding environment, we can identify anomalous tags by collecting IDs of the normal tags and considering the remaining tags as anomalous. To select normal tags within a range, we replace the action $-B$ with $-A$ in line 10 of Alg. 2. This sets all normal tags in Γ_i to state B, while the other tags remain in state A. Subsequently, we issue a query command to inventory the tags with flag B. However, this approach requires additional select commands for each range, which is not optimal. Instead, we can construct a new mask by combining the current group string mask $maskG$ with the $mask_i$ and use it to select normal tags. For the first mask, the modified `Select` command becomes:

$$\text{Flag} \leftarrow BA : \mathcal{S}(2, 4, 3, p', len_G + len_1, maskG + mask_1). \tag{15}$$

The modified algorithm is shown in Alg. 3. In comparison to Alg. 2, the new mask string is $maskG + mask_i$ (shown in Lines 3, 5, and 8). Furthermore, in line 10, the query command is modified to read the tags with flag B, making it $Q(0, 2, 1)$. This modification removes the need for an additional select command to silence tags from other ranges. As a result, selecting normal tags within a range is now achieved with one less select command, leading to improved time efficiency by reducing the total number of select commands by m.

ALR-F800


Alien 9662
Alien 9440
Higgs9

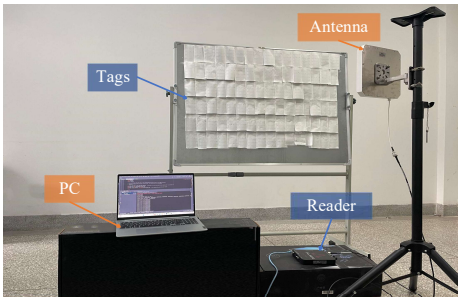

Fig. 3. Experimental setup.

IV. EVALUATION

A. Experimental Setup

As shown in Fig. 3, we evaluate our protocols using the Alien ALR-F800 commodity RFID reader, an Enterprise class reader capable of bit-level masking, which is essential for our protocol. The bit-level mask conforms to C1G2 specifications; the disability of other commodity readers might be due to they support certain functions internally while not exposing these detailed functionalities to users. The reader is connected to a directional antenna that is with 9 dBic gain, operating at around 920 MHz. In the experiments, we deploy 100 to 500 commodity tags (Alien 9662 and Alien 9440 with Higgs9 chipset) to evaluate the protocols' performance.

We program all tags by writing the corresponding data into them in advance. This includes writing normal tags with data within their normal range and anomalous tags with data beyond their normal range. The programming process is achieved using the `Write` command, specified by C1G2. The `Write` command comprises three fields: `MemBank`, `WordPtr`, and `Data`. `MemBank` determines the memory bank to be written, commonly set to MemBank-3 (see Section III-A). `WordPtr` specifies the word-level address for the memory write, where each word is 16 bits long. `Data` contains the user information to be written. Using this command, commercial readers can write specific information into the memory of RFID tags.

B. Time and Space Efficiency of Encoding

In this subsection, we evaluate the number of selects and the length of encoding vector of our protocols via simulations. We study the number of selects for LS (US is the same as LS and LUS can get similar conclusion). In Fig. 4, we vary the size of τ from $1 \times 10,000$ to $20 \times 10,000$ at the step of 1000, i.e., $\tau = 10,000, 11,000, \ldots, 20,000$. We compare the average number of selects in interval $[1, \tau]$ with RQ and EnRQ, respectively. As we can see, the number of selects of ATI, RQ and EnRQ all experience rise as τ increases because the probability of non-zero bits in their encoding vectors increases as τ increases. The average number of selects for ATI is only one select more than EnRQ10. Moreover, RQ requires more selects than both ATI and EnRQ.

In Fig. 5, we study the length of encoding vector of ATI, RQ, and EnRQ with respect to τ, where the values are the same as in Fig. 4. It is evident that the length of the encoding vector of ATI is close to RQ and EnRQ4, and much shorter than EnRQ16 and EnRQ10. The length of the encoding vector

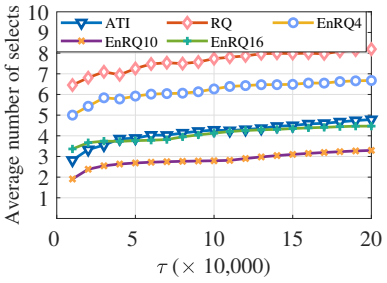

Fig. 4. Average number of selects.

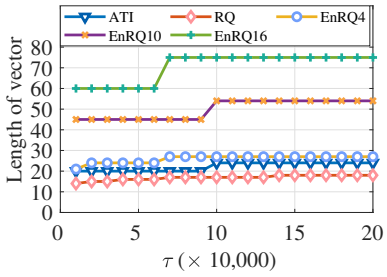

Fig. 5. Encoding length.

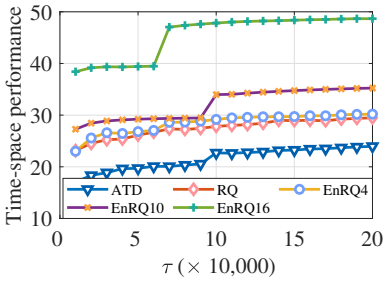

Fig. 6. Time-space efficiency.

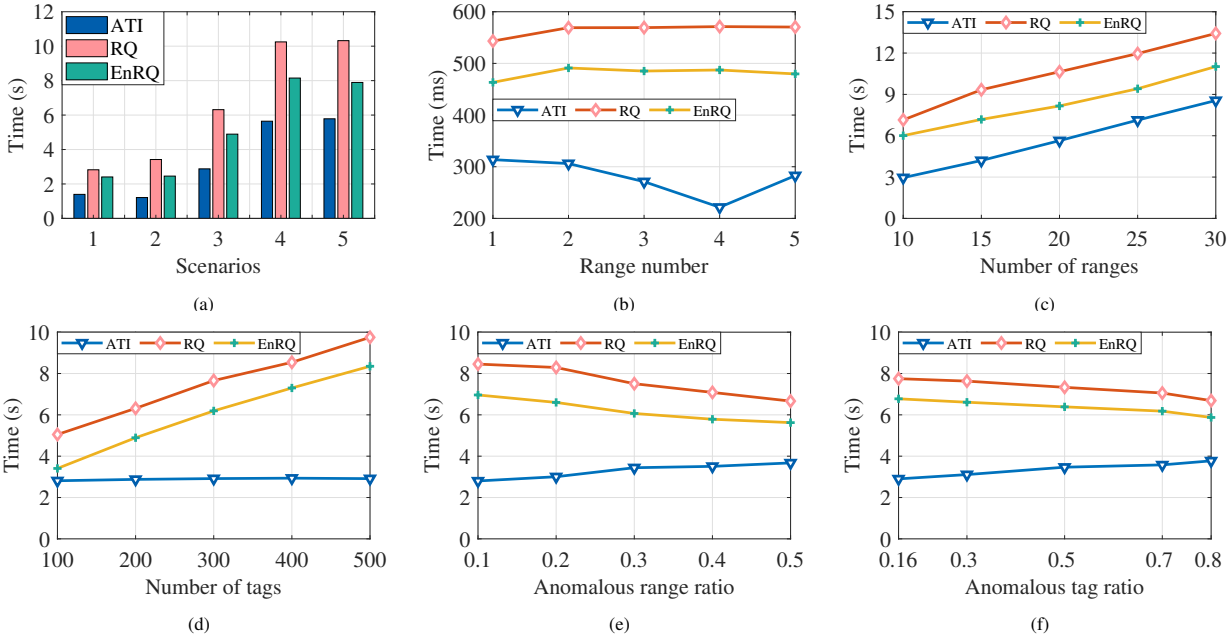

Fig. 7. Comparison of time efficiency.

of RQ is similar to ATI's, but the number of selects for RQ is approximately double that of ATI's, which is consistent with the results in Fig. 4. Furthermore, although ATI requires one more select on average than EnRQ10, its encoding vector's length is less than half of EnRQ10's.

To provide a more intuitive view of the time-space efficiency of all protocols, we devise an evaluation metric as:

$$P_{ts} = \alpha \times w1 \times n_{selects} + w2 \times L_{vector}, \tag{16}$$

where $w1$ and $w2$ are the weights of the number of selects and the encoding length, respectively, both set to 0.5. We set $\alpha = 5$ to balance the order of magnitude based on the analysis from Fig. 4 and Fig. 5. From Fig. 6, it is evident that ATI achieves a favorable trade-off between the number of selects and the length of the encoding vector, which corresponds to time and space efficiency. For example, when $\tau = 20,000$, the metric of ATI is 18.3, RQ is 24.5, and EnRQ16 is 39.2.

C. Time Efficiency

We now study the execution time of ATI in a commodity RFID system. We adapt RQ [20] and EnRQ [21] to support multi-range queries, and take them as the baselines for comparison. For EnRQ, we use EnRQ16 as the baseline, as it is the most time-efficient compared to EnRQ10 and EnRQ4. In each experiment, we randomly choose $2m$ numbers between 1 and 10,000 as boundaries for m ranges. This allows us to cover a wide range of real-life situations, as the difference between the upper and lower limits of a given data range is generally not significantly large. Afterward, we get the execution time of ATI, RQ, and EnRQ by averaging the results of 20 runs.

In Fig. 7(a), we compare the execution time of ATI, RQ, and EnRQ across five scenarios. In scenario 1, we set $m = 5$, $n = 100$, $p_m = 0.4$, $p_n = 0.1$. Here, m is the number of ranges, n is the number of tags, p_m is the ratio of the number of anomalous ranges containing anomalous tags to the total number of ranges, and p_n is the ratio of the number of anomalous tags in each anomalous range to the total number of tags in that range. Scenario 2, same parameters as scenario 1, but tags are programmed with data having trailing zero boundaries (1000 to 10,000). Other experiments use $2m$ randomly chosen boundaries. In scenario 3, we double m and n, which maintains the number of tags in each range, and set $p_m = 0.2$, $p_n = 0.2$. In scenario 4, we deploy more tags, i.e., $n = 300$, $m = 20$, $p_m = 0.1$, while keeping the number of anomalous ranges, and set $p_n = 0.2$. In scenario 5, we double p_m and p_n. To examine the execution time of all protocols, we take a closer look at scenarios 2 and 5. Compared to scenario 1, in scenario 2, the execution time of ATI decreases for it requires fewer selects for its encoding vector is tailored for the trailing zero boundaries. In scenario 5,

the modified RQ and EnRQ take 10.32s and 7.89s to conduct the multiple range detection, respectively. Our protocol, ATI, reduces the time to 5.78s by leveraging the fixed-encoding and grouping method, which is 44.1% and 26.7% faster than RQ and EnRQ, respectively. Similar conclusions hold true for the other three scenarios, demonstrating the superiority of ATI over the baselines in terms of time efficiency.

In Fig. 7(b), we illustrate the average time of all protocols to identify each range in scenario 1. ATI requires more time to identify the first and second anomalous ranges compared to the other three normal ranges. The fluctuation is due to the varying number of selects required for different ranges. RQ and EnRQ show consistent times for each range, as they collect all normal tags within each range.

Now, we study the impact of the number of ranges, number of tags, anomalous range ratio, and anomalous tag ratio on the execution time of our protocols and the baseline. In Fig. 7(c), we study the effect of the number m of ranges on the execution time. With $n = 300$, $p_m = 0.1$, and $p_n = 0.2$, we vary m from 10 to 30 with a step of 5. The execution time of ATI, RQ, and EnRQ increases as m increases since each is processed separately. RQ and EnRQ show slower growth due to reduced tags per range; ATI exhibits roughly linear growth.

In Fig. 7(d), we study the execution time of ATI, RQ, and EnRQ with respect to the number n of tags. Keeping $m = 10$, $a_m = 2$, and $a_n = 2$ fixed, we vary n from 100 to 500 with a step of 100; where a_m is the number of anomalous ranges and a_n is the number of anomalous tags in each range. The baseline protocols show increasing execution time with more tags, while our protocols remain stable. That is because, with the given values of m, $a_m = 2$, and $a_n = 2$, the number of normal tags increases as n increases. RQ and EnRQ collect all normal tags in each range, leading to an increase in execution time. In contrast, ATI only collects anomalous tags, maintaining a stable execution time with fixed numbers of anomalous ranges and tags in each range.

In Fig. 7(e), we show how the anomalous range ratio p_m influences the execution time. We fix $m = 10$, $n = 300$, $p_n = 0.2$ and vary p_m from 0.1 to 0.5 with a step of 0.1. The execution time of RQ and EnRQ decreases with the anomalous range ratio, while our protocols exhibit an increasing trend as the anomalous range ratio rises. The reason is that our protocol requires more time to collect more anomalous tags as p_m increases, while the baseline collect fewer normal tags.

Fig. 7(f) shows how the execution time is affected by the anomalous tag ratio p_n. With $m = 10$, $n = 300$, and $p_m = 0.1$ fixed, we vary a_n from 5 to 25 (step of 5), resulting in p_n ranging from approximately 0.16 to 0.8. The execution time of RQ and EnRQ decreases due to fewer normal tags collected in anomalous groups, while the execution time of ATI increases with p_n as more anomalous tags are collected. It consistently outperforms the baselines in terms of time efficiency, especially when p_n is small. For example, when p_n is 0.16, the time of RQ and EnRQ is 7.75s and 6.77s, respectively; while ATI takes only 2.9s, representing a 1.33× increase compared to EnRQ.

V. Related Work

Multi-range query is to identify tags with data beyond their normal data range. The basic solution is to collect all tags' information and check whether their data falls within their expected range. In recent years, many advanced protocols have been developed to enhance the time efficiency of information collection [22]–[25], [27]. Chen et al. [23] maximize the useful single slot by using multiple hash functions to reconcile the collision slots. Qiao et al. [27] introduce a tag-ordering polling protocol with partitioned Bloom filters, which effectively reduces energy consumption at a minimal cost to time efficiency. Xie et al. [25] design a data structure called Minimal Perfect Hashing based Filter (MPHF) to filter out the irrelevant tags and avoid the interference with the target tags. Liu et al. [22] propose an incremental polling protocol (IPP) that drops the polling vector to just 1.6 bits long. These methods provide viable solutions for multi-range query, but they have limitations: collecting all tags can be time-consuming, especially when the number of anomalous tags is small compared to the total tags; they are not compatible with the EPCglobal Class1 Gen2 (C1G2) standard, making them unsuitable for commodity RFID systems.

The range query (RQ) approaches [20], [21] check whether there are any tags with data between a lower and upper boundary and are compatible with C1G2. They perform well in single-range scenarios by dividing tags into anomalous and normal sets and then querying the anomalous set to identify the anomalous tags. However, in multi-range scenarios, the anomalous set may include anomalous and normal tags from different ranges, leading to false alarms even when there are no anomalies in the targeted range.

VI. Conclusion

This paper focuses on multi-range query in a commodity RFID system, which aims to identify tags with data beyond their expected data range. We design one tailored solution called anomalous tag identification (ATI). In ATI, we build a fixed-length encoding vector to incorporate grouping information, effectively separating anomalous tags associated with a range from tags of other ranges. Extensive experimental results demonstrate that our protocol ATI improves the time efficiency of anomalous tag identification when compared with the baselines under different parameters.

Acknowledgment

This research is financially supported by the Fundamental Research Funds for the Central Universities (Nos. B240201062 and B240201064), the National Natural Science Foundation of China (Nos. 62102079, 62332013, 62180005, and 62306104), the Natural Science Foundation of Jiangsu Province under Grant (No. BK20230949), Hong Kong Scholars Program (No. XJ2024010), Research Grants Council of the Hong Kong Special Administrative Region, China (GRF Project No. CityU11212524), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

REFERENCES

[1] J. Zhang, X. Liu, S. Chen, X. Tong, Z. Deng, T. Gu, and K. Li, "Toward robust RFID localization via mobile robot," *IEEE/ACM Transactions on Networking*, vol. 32, no. 4, pp. 2904 – 2919, 2024.

[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] V. C. Maia, K. M. de Oliveira, C. Kolski, and G. H. Travassos, "Using RFID in the engineering of interactive software systems: A systematic mapping," in *Proc. of ACM HCI*, 2023, pp. 1–37.

[4] S. Zhang, Z. Ma, K. Lu, X. Liu, J. Liu, S. Guo, A. Y. Zomaya, J. Zhang, and J. Wang, "Hearme: Accurate and real-time lip reading based on commercial RFID devices," *IEEE Transactions on Mobile Computing*, vol. 22, no. 12, pp. 7266–7278, 2022.

[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, X. Chen, S. Chen, X. Liu, Y. Wang, and L. Chen, "TagSheet: Sleeping Posture Recognition with an Unobtrusive Passive Tag Matrix," in *Proc. of IEEE INFOCOM*, 2019, pp. 874–882.

[7] Y. Wang and Y. Zheng, "TagBreathe: Monitor Breathing with Commodity RFID Systems," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 969–981, 2020.

[8] X. Liu, B. Zhang, S. Chen, X. Xie, X. Tong, T. Gu, and K. Li, "A wireless signal correlation learning framework for accurate and robust multi-modal sensing," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 9, pp. 2424 – 2439, 2024.

[9] W. Gong, H. Wang, S. Li, and S. Chen, "Glac: High-precision tracking of mobile objects with cots RFID systems," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2331 – 2343, 2024.

[10] B. Liang, P. Wang, R. Zhao, H. Guo, P. Zhang, J. Guo, S. Zhu, H. H. Liu, X. Zhang, and C. Xu, "Rf-chord: Towards deployable RFID localization system for logistic networks," in *Proc. of USENIX NSDI*, 2023, pp. 1783–1799.

[11] X. Liu, B. Zhang, L. Wang, S. Chen, X. Xie, X. Tong, T. Gu, and K. Li, "Fine-grained recognition of manipulation activities on objects via multi-modal sensing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9614 – 9628, 2024.

[12] J. Liu, S. Chen, M. Chen, Q. Xiao, and L. Chen, "Pose sensing with a single RFID tag," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2023–2036, 2020.

[13] S. Li, S. Li, M. Chen, C. Song, and L. Lu, "Frequency scaling meets intermittency: Optimizing task rate for RFID-scale computing devices," *IEEE Transactions on Mobile Computing*, vol. 23, no. 2, pp. 1689–1700, 2023.

[14] X. Liu, Y. Huang, Z. Xi, J. Luo, and S. Zhang, "An efficient RFID tag search protocol based on historical information reasoning for intelligent farm management," *ACM Transactions on Sensor Networks*, 2023.

[15] J. Su, Z. Sheng, C. Huang, G. Li, A. X. Liu, and Z. Fu, "Identifying rfid tags in collisions," *IEEE/ACM Transactions on Networking*, vol. 31, no. 4, pp. 1507–1520, 2022.

[16] K. Liu, L. Chen, J. Yu, and H. Cui, "On batch writing in cots rfid systems," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 3846–3857, 2023.

[17] X. Liu, J. Cao, Y. Yang, W. Qu, X. Zhao, K. Li, and D. Yao, "Fast RFID sensory data collection: Trade-off between computation and communication costs," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1179–1191, 2019.

[18] J. Wang, O. Abari, and S. Keshav, "Challenge: RFID hacking for fun and profit," in *Proc. of ACM MobiCom*, 2018, pp. 461–470.

[19] Y. Bu, L. Xie, Y. Gong, J. Liu, B. He, J. Cao, B. Ye, and S. Lu, "Rf-3dscan: RFID-based 3d reconstruction on tagged packages," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 722–738, 2021.

[20] J. Liu, X. Yu, X. Liu, X. Chen, H. Liu, Y. Wang, and L. Chen, "Time-efficient range detection in commodity RFID systems," *IEEE/ACM Transactions on Networking*, vol. 30, no. 3, pp. 1118–1131, 2022.

[21] X. Yu, J. Liu, S. Zhang, X. Chen, X. Zhang, and L. Chen, "Encoding-based range detection in commodity RFID systems," in *Proc. of IEEE INFOCOM*. IEEE, 2022, pp. 680–689.

[22] 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.

[23] 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.

[24] Y. Qiao, S. Chen, T. Li, and S. Chen, "Tag-Ordering Polling Protocols in RFID Systems," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 1548–1561, 2016.

[25] X. Xie, X. Liu, K. Li, B. Xiao, and H. Qi, "Minimal Perfect Hashing-Based Information Collection Protocol for RFID Systems," *IEEE Transactions on Mobile Computing*, vol. 16, no. 10, pp. 2792–2805, 2017.

[26] "EPC radio-frequency identity protocols generation-2 UHF RFID standard," GS1, ISO/IEC 18000-63, Jul. 2024, https://www.gs1.org/standards/epc-rfid/uhf-air-interface-protocol.

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