Missing-Tag Detection With Presence of Unknown Tags

Youlin Zhang* Shigang Chen* You Zhou*† Olufemi Odegbile*
*Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL 32611, USA,
†Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA, USA
Email: {youlin, sgchen}@cise.ufl.edu, {youzhou}@google.com, {oodegbile}@cise.ufl.edu

***Abstract*—Radio Frequency Identification (RFID) technology has been proliferating in recent years, especially with its wide usage in retail, warehouse and supply chain management. One of its most popular applications is to automatically detect missing products (attached with RFID tags) in a large storage place. However, most existing protocols assume that the IDs of all tags within a reader's coverage are known, while ignoring practical scenarios where the IDs of some tags may be unknown. The existence of these unknown tags will introduce false positives in those protocols, degrading their performance. Some prior art studies this problem, but their time efficiency is low, especially when the number of unknown tags is large. In this paper, we propose a missing tag detection protocol based on compressed filters, which not only reduces the filter size for better time-efficiency but also helps dampen the interference of unknown tags for high missing-tag detection accuracy. To further improve the performance, we propose a new way for tags to report their presence, greatly reducing collisions and thus improving the detection probability. Extensive simulations demonstrate that our compressed filter and collision-reduction method reduce the protocol execution time by 83% to 92% under the same missing-tag detection probability, when comparing with the best prior work.**

I. Introduction

In recent years, RFID technologies have been proliferating, with numerous applications that have been developed including supply chain management [5]–[7], [12], [14], [15], [17], [20], [23], [24], [26], [27], [31], [36]–[38], object tracking [30], [32], [33], theft prevention [13], [16], [18], [25], [29], [34], [35] and so on [11], [22]. In an RFID system, objects are attached with tags, each having a unique ID, which can be identified by a reader that is connected with one or many antennas deployed to monitor tags within a coverage area and collect statistics. Comparing with traditional barcodes which are read by laser scanners with line of sight in a very short distance, RFID technologies have great advantages that they can be read wirelessly over a longer distance without line of sight and that they are capable of performing simple computations.

One of the most popular applications using RFID tags is to automatically detect missing products in a storage place. According to [3], [28], shoplifting, employee theft and vendor fraud have become the major causes of lost capital for retailers like Wal-Mart. In practice, we may have someone walk around to check and count items. This is not only laborious but also error-prone, considering that the products may be stacked together, goods on racks may need a ladder to access and they may be blocked behind columns. However, if we attach each item with an RFID tag, the whole detection process can be automated with an RFID reader communicating with tags to check whether any of them are missing.

In general, the missing tag detection problem can be classified into two types: deterministic detection [13], [16], [35] and probabilistic detection [18], [25], [29], [34]. The deterministic protocols identify exactly which tags are missing, while the probabilistic ones report a missing tag event with a certain detection probability. Usually a probabilistic detection protocol runs faster with smaller overhead while a deterministic one gives stronger results with larger overhead. In practice, these two types of missing tag detection protocols are complementary to each other and can be used together to achieve better results. For example, a probabilistic detection protocol can be scheduled frequently to detect a missing tag event. Once it detects some missing tags, a deterministic protocol can be executed to find out exactly which tags are missing.

Most existing protocols assume that the IDs of all tags in the coverage area are known, while ignoring practical scenarios where the IDs of some tags may be unknown. Consider an airport that deploys RFID technologies to monitor the passenger baggages that belong to different airline companies, with each passenger baggage attached with an RFID tag. Every airline company will want to detect whether baggages of its passengers are missing. However, in an area where baggages of other companies are present, we will face the problem of missing-tag detection for tags (baggages) of one airline, with the presence of unknown tags from other airlines that are of no interest but cause interference. In a more general warehouse setting, many clients rent space to store their products, which are tagged. Suppose a client deploys an RFID reader and antennas to detect whether some of its products are missing. Nearby tags on the products of other clients will respond to the reader if they happen to be within the coverage area. These tags are unknown to the client that performs missing-tag detection.

This paper focuses on probabilistic missing tag detection and considers a problem of practical significance: detecting missing-tag events with presence of unknown tags. Many traditional methods such as [18], [19], [29] cannot handle unknown tags. It is shown in [25] that the presence of unknown tags will introduce false positives and compromise the detection accuracy. The most related work is [25], [34], which can detect missing-tag events with unknown tags. The efficiency of [25] drops greatly in the presence of a large number of unknown tags because it does not filter out the unknown tags in its operations. The performance of [34] is much better, thanks to its use of Bloom filters to remove the unknown tags, but it requires tags to implement a large number of hash functions (which may be impractical). More importantly, its Bloom filter design and in particular its use of Bloom filter to communicate from tags to the reader are costly and result in long execution time. Reducing execution time is important for large RFID systems with numerous tags operating in low-rate communication channels. This is particularly true in a busy warehouse environment [5], [7], [20], [23], [24], [37], [38], where we want to minimize disruptions caused by RFID protocol execution to normal warehouse operations.

In this paper, we propose a new protocol that can achieve reliable and time-efficient missing tag detection with the presence of unknown tags. Our idea is to design a new type of compressed filter that can work with any number of hash functions (as low as one) available to the tags, yet with a decreased filter size for better efficiency than the prior art. More importantly, unknown tags are removed by the filter without the need to decompress the filter. Tags perform lookup directly on the compressed filter as it is received. The new filter is suitable for resource-limited tags because it does not use generic compression algorithms, but requires only simple operations. To further improve the performance, we combine sampling and multi-hashing for tags to report their presence, greatly reducing collisions.

The main contributions of this paper are summarized as follows: First, we provide an efficient solution to the missing-tag detection problem under more general scenarios where unknown tags are present.

Second, we propose a compressed filter that is suitable for RFID systems, with low overhead and simple operations, allowing tags to perform membership check without decompressing the filter. The filter size is smaller than the optimal Bloom filter used by the prior art.

Third, we formally analyze the performance of our protocol.

Finally, we conduct extensive simulations to complement our theoretical analysis and evaluate the performance of our protocol. The results demonstrate that our compressed filter and collision-reduction method reduce the protocol execution time by 83% to 92% under the same missing-tag detection probability, when comparing with the best prior work.

II. System Model and Problem Definition

A. System Model

Consider a large RFID system, where each object is attached with an RFID tag. Each tag has a unique ID by which we can identify an object, and it is capable of performing certain computations. An RFID reader is deployed in the system to monitor the tags within its coverage and interrogates with the tags using backscattered signal in a frame-slotted ALOHA protocol. We also assume that the reader has access to a backend server which stores the tag IDs of interest.

In one communication round, the reader will first initialize the communication by broadcasting a request that includes all necessary parameters such as a frame size and random seeds. Each tag after receiving the request performs some calculations and decides which slot(s) it will respond. The request is followed by an ALOHA frame consisting of f slots, in which tags can transmit their responses. Based on the number of tags that respond in each slot, the time slots are classified in two types: *empty slots*, where no tag responds, and *busy slots*, where one or multiple tag respond. A busy slot can be further classified into two types: a *singleton slot*, where only one tag responds, and a *non-singleton slot*, where multiple tags respond. The reader monitors the status of each slot and converts the time frame into a bit array, zero for each empty slot and one for each busy slot.

B. Problem Definition

Let E be the set of tags attached to all objects in an RFID system and T be a subset of E, that consists of tags we want to monitor and whose IDs are known. $T \subseteq E$. We denote the cardinality of T as n and the set of $E-T$ as U, which consists of tags whose IDs are unknown. Note that tags in T and U can all respond to the RFID reader in our system and there is no difference with respect to their operations among tags in these two sets except that the IDs of tags in T are known and of our interest while the IDs of tags in U are unknown. In the sequel, we will refer tags in U as *unknown tags* and tags in T as *known tags*.

The problem of missing tag detection with presence of unknown tags is to detect whether any tag(s) in T is missing with the presence of tags in U. The requirement is that the probability for detecting a missing tag event after one protocol execution is at least α if M or more tags are missing, where $0 < \alpha \leq 1$, $M \geq 1$, which are two system parameters set by users based on their practical need. We shall report a missing tag event with a probability α if $m \geq M$, where m is the number of tags in T that are missing. If the number of tags that are missing in T is smaller than M, we can still detect a missing tag event, with a detection probability smaller than α.

As an example, if we set $\alpha = 95\%$ and $M = 10$, after an execution of our protocol we shall be able to detect an event of missing 10 or more tags with at least 95% probability. If the protocol is executed w times, the detection probability will be at least $1-(1-\alpha)^w$, which will approach to 100% as w increases. In this way, a missing tag event will eventually be detected overtime no matter what the values of α and M are [8], [9].

C. Performance Metrics

In this section, we describe the metrics for evaluating the performance of a missing tag detection protocol.

1) Execution Time: As is stated previously, RFID systems operate in low-rate communication channels. To apply such protocols in a busy warehouse environment, it is desirable that the execution time can be reduced as much as possible, especially when the number of tags is very large.

2) Detection Probability: The probability of detecting a missing tag event is another important performance metric. The detection requirement is defined in Section II-B. In practice, we want the detection probability α to be close to 1.

III. Protocol Design

Our protocol for missing tag detection with presence of unknown tags consists of two phases: unknown tag filtration and missing tag detection. In phase one, since the reader knows the tag IDs in T, it can construct a filter that encodes the membership of all tags in T and use this filter to filter out unknown tags in U. Rather than using the original Bloom filter, we adopt a segment design and implement an algorithm that works on simple RFID tags to compress the Bloom filter into a compressed filter. Through experiments, we show that our compressed filter can achieve a lower false positive ratio than [34] with a more compact space. In phase two, we combine sampling and multi-hashing to reduce the tags' collisions in communication, which further improve the time efficiency. In the following, we describe in detail our protocol for missing-tags detection with presence of unknown tags.

A. Prior Work

The best prior work is BMTD [34] which uses a design based on Bloom filters [10], a compact data structure that is used to encode a set $T = \{t_1, t_2, ..., t_n\}$ of n elements (tags in our context). A Bloom filter is a bitmap of f bits. Each tag is encoded by mapping the tag ID to k bits using k independent hash functions $h_1, h_2, ..., h_k$ and setting those bits to one. To check whether a tag is a member in T, we hash it to k bits in the filter and check whether these bits are all 1's. There is no false negative in a Bloom filter: A tag in T will always pass the membership check. However, there is *false positive*: A tag not in T may also pass the membership check. The false positive ratio is given by [10]:

$$P_{fp} = (1 - (1 - \frac{1}{f})^{kn})^k \approx (1 - e^{-kn/f})^k. \tag{1}$$

Given the size n of T and the number k of hash functions, BMTD uses the following optimal size f for its Bloom filter, with its false positive ratio also given below.

$$\begin{aligned} f &= \frac{nk}{\ln 2}, \\ P_{fp} &= (\frac{1}{2})^k. \end{aligned} \tag{2}$$

The RFID reader will broadcast the above filter that encodes the tags in T. Upon receipt of the filter, the unknown tags will filter themselves out if they cannot pass the membership check. The optimal Bloom filter may need to be broadcast multiple times in order to achieve a desirable false positive ratio.

The k hash functions used in each Bloom filter must be independent, and the hash functions of different filters must also be independent. For example, in order to achieve a positive ratio of 0.001, there will need 10 independent hash functions, which is a burden for simple hardware of a tag. If a tag can only afford a fewer number k of hash functions, we cannot use optimal filters as BMTD does, whose false positive ratio is limited to $(\frac{1}{2})^k$. We have to use non-optimal filters.

An optimal filter has 50% zeros, and therefore its false positive ratio is $(\frac{1}{2})^k$. In (1), we can reduce P_{fp} to an arbitrarily small number by increasing the filter size f. By increasing f, we increase the ratio ρ of zeros in the filter, and the false positive ratio becomes ρ^k, which can be made arbitrarily small when ρ is driven down.

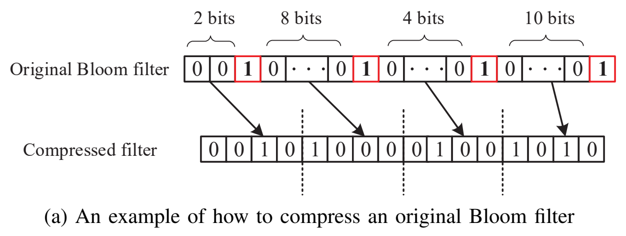
The problem is that a larger filter takes more time for the reader to broadcast, which degrades time efficiency. Fortunately, if the filter contains a large number of zeros, we can compress it to a smaller size. In fact, according to [21], the size of the compressed filter can be smaller than the size of the optimal filter with the same false positive ratio. However, there arise two different problems. First, tags are resource-limited and cannot perform generic compression and decompression algorithms such as Huffman compression and LZ compression [1]. Second, more importantly, tags do not have the memory to hold the decompressed filter (with its original size) to perform membership check. Therefore, we need to design a compressed filter to work with limited resources. Moreover, tags should be able to directly work with the compressed filter for membership check without decompression. The operations for tags should be simple.

B. Phase One: Unknown Tag Filtration by Compressed Filter

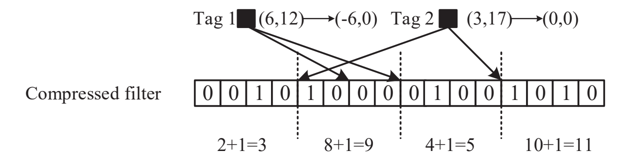
To reduce the interference of unknown tags, the reader uses a compressed filter with a small pre-set false positive ratio p_1 such that the majority of unknown tags are filtered and do not participate in phase two. We design a compressed filter that can work with any value of k_1 (as low as one) hash functions, achieve any specified false positive ratio p_1, require simple operations by tags, and perform membership check without decompression.

Given the values of p_1 and k_1, the reader constructs a Bloom filter whose size is determined by (1). The smaller the value of k_1 is, the larger the value of f_1 will be, and the more the number of zeros will be in the filter, which gives opportunity for compression, allowing the reader to broadcast a smaller filter to tags and save execution time.

For compression, the reader divides the Bloom filter into segments of consecutive zeros that are separated by the bits of ones. We replace each segment of *consecutive zeros* by the *number* of zeros in the segment. For example, a segment of 20 zeros is replaced by the number 20, which compresses 20 bits to a number l_1 of bits that represents 20 in binary. Different segments may have different numbers of zeros. The reader finds the maximum number L_{max} of zeros in all segments. It sets $l_1 = \lceil \log_2(L_{max} + 1) \rceil$. The reader converts each segment of zeros to an l_1-bit number, and converts the whole Bloom filter into a sequence of l_1-bit numbers, which form the compressed filter. The bits of ones in the original filter are



(a) An example of how to compress an original Bloom filter



(b) An example of how to use a compressed filter for membership check

Fig. 1: An illustration of how to compress an original Bloom filter and use a compressed filter for membership check.

implied in the compressed filter, one such bit between any two consecutive segments.

The reader broadcasts a request with parameters including the size of the filter. It then transmits the compressed filter to all tags. If the filter is too long, the reader may divide it into parts and transmit each part in a time slot. For example, if we use time slots of size for transmitting 96-bit tag IDs, each slot can carry $\frac{96}{l_1}$ segments.

For membership check, each tag maps itself to k_1 bits in the original Bloom filter by k_1 hash functions. It then checks whether those k_1 bits are all ones. However, it does not possess the original Bloom filter, but only receives the compressed filter from the reader. For each of the k bits, the tag loads the hash position of the bit in the filter into a counter. As it receives the compressed filter from the reader, for each l_1-bit number received, it subtracts the number plus one from the counter, where the number represents a segment of consecutive zeros and the one represents a bit 1 between two segments. This process continues until the counter is reduced to zero or a negative number. If the counter is reduced to zero, it means that the bit that the tag is mapped to in the original Bloom filter is located right between two segments of zeros; that bit must be one. If the counter becomes negative, it means that the bit that the tag is mapped in the original Bloom filter must be zero. There are k_1 counters for the tag. If all k_1 counters are reduced to zeros, it means that the tag has passed the membership check and it is claimed to be a member in T. If any of the counters becomes negative, the tag fails the membership check. All tags that pass the check will participate in phase two. All tags that fail the check, including the majority of unknown tags in U, are filtered out and will stay silent in phase two.

Fig. 1 shows an example of how to compress an original Bloom filter and how to use a compressed filter for membership check. In Fig. 1a, the upper bit array is the original Bloom filter and the lower array is the resulting compressed filter. As we can see, there are four ones in the original Bloom filter, and thus the compressed filter has four segments. The numbers of zeros in those segments are 2, 8, 4 and 10 respectively. We have $L_{max} = 10$ and $l_1 = \lceil \log_2(L_{max} + 1) \rceil = 4$. The compressed filter consists of 2, 8, 4 and 10 in 4-bit binary format. The compression ratio is $\frac{28}{16} = 1.75$. Fig. 1b shows an example of how to perform membership check using the compressed filter. Let $k_1 = 2$. The hash values of two tags are (6, 12) and (3, 17) respectively. The expression below each l-bit number is the sum of the number and 1. When performing membership check, each tag will subtract these sums from the hash values until the results are non-positive. As we can see, for tag 1, its hash values will be subtracted to (-6, 0), while for tag 2, they are subtracted to (0, 0). Thus, tag 2 passes the membership check and tag 1 does not.

C. Motivation for Phase Two

After phase one filters out most of unknown tags, we use phase two for the remaining tags (particularly those in T) to report their presence to the RFID reader. In BMTD [34], these tags together transmit a Bloom filter to the reader, with each tag encoded as k_1 ones in the filter. In order to achieve a low false positive ratio, the filter size has to be large. For example, by (1), if $k_1 = 3$, the filter must be at least $12.4n$ bits to ensure a false positive ratio of 0.01, which means 12.4 bits per encoded tag, where n is the total number of tags.

The time for delivering each bit in the filter to the reader can be costly. In practice, it is difficult for numerous tags to synchronize their transmissions at bit level, especially for a long filter of many bits. Many prior works resort to more robust designs with one time slot delivering a single bit [7], [12], [15], [17], [23], [24], [27], [31]. We have implemented such a design that conforms to the EPC C1G2 standard [2] following [4], which works as follows: The reader initiates a time frame of a specified number of slots. Each tag chooses a random slot to transmit its ID, and the slot structure is QueryRep $\rightarrow$ RN16 $\rightarrow$ ACK $\rightarrow$ tag ID, where QueryRep is a command transmitted by the reader to start a slot, RN16 is a 16-bit random number transmitted by a tag for collision detection, ACK is transmitted by the reader and carries the received RN16, and ID is transmitted by the tag, which is followed by another QueryRep from the reader to start the next slot. We have reconfigured the reader to skip ACK/tag ID and immediately send QueryRep after RN16 to start the next slot. In this way, each slot does not carry any ID but instead just one bit information: If any tag transmits RN16, the reader observes a busy channel and registers a bit 1; if no tag transmits, the reader sees an idle channel (empty slot) and registers a bit 0. The time frame of slots will thus be turned into a bitmap. In case of BMTD [34], if each tag transmits in k_1 randomly chosen slots, the bitmap is a Bloom filter encoding the tags. The slot structure is now QueryRep $\rightarrow$ RN16, denoted as T_{short}. T_{short} translates into 2.69ms based on the settings in [4]. In comparison, the original slot structure for tag ID, denoted as T_{ID}, takes 14.37ms.

BMTD [34] is inefficient because it requires many bits

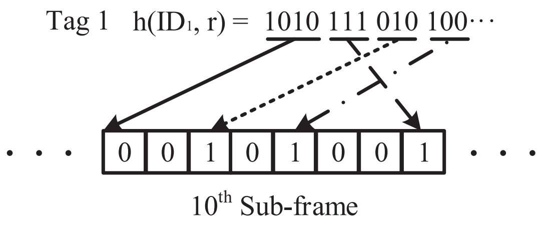


Fig. 2: An illustration of multi-hashing, where $k_2 = 3$ and $l_2 = 8$. The first four bits of $h(ID_1, r)$ are used to decide in which sub-frame (the 10th in this example) the tag will transmit. The first three bits after those four bits are used to decide the first mapped slot of this tag in this sub-frame. The next three bits are used to decide the second mapped slot, and the following three bits are used to decide the third mapped slot.

(slots) per tag. We argue that the minimum number of slots needed to report the presence of a tag is actually one. In our design, a tag not filtered in phase one will transmit at most once, using a single slot. Suppose we assign each tag to a single slot by hashing. The slots can be classified into three types: singleton slots, collision slots, and empty slots, which have a single tag, multiple tags, and zero tag in T, respectively. The reader knows the IDs of tags in T, and thus it can predict which slots are singletons, which have collisions, and which are empty. Monitoring the status of the slots, if the reader observes that an expected singleton/collision slot turns out to be empty, it knows that the tags in T that are supposed to transmit in this slot must be missing and thus it detects a missing-tag event successfully. For now, we will ignore the unknown tags that pass phase one; they may cause noise to make detection fail probabilistically. Their number is small, and our analysis will consider the noise that they introduce into our detection and make sure that the accuracy requirement will be met under such noise.

Among the three types of slots, singletons are most productive. If a tag in a singleton slot is missing, the slot will become empty and the reader will detect a missing tag. For a collision slot, two or more tags must be all missing in order for the slot to become empty, which happens with a smaller probability. We want to design our solution that maximizes the number of singleton slots in order to improve detection probability, while reducing the number of empty slots in order to improve time efficiency. To do so, we map each tag to multiple slots and choose the one that makes a singleton for the tag to transmit.

Moreover, to make the solution more time efficient, we sample the tags so that only a portion of them will report their presence. This is fine for probabilistic detection as long as the sampling probability is large enough to meet the accuracy requirement of missing-tag detection.

D. Phase Two: Missing Tag Detection

In phase two, the reader first samples the tags in T and constructs a bitmap that is expected to receive from these sampled tags if all of them are present and there is no unknown tag (noise). Let the size of the bitmap be f_2. The bitmap will be collected by using a time frame of f_2 slots, which is divided into sub-frames of l_2 slots each, where l_2 is divisible by f_2. The reader knows the IDs of tags in T. For each tag, it performs a hash $h(ID, r)$, where ID is the tag's ID and r is a random seed. The reader uses the first $\log_2 \frac{f_2}{l_2}$ hash bits to map the tag to a sub-frame, and uses the following hash bits to further map the tag to l_2 slots in the sub-frame, each slot taking $\lceil \log_2(l_2) \rceil$ hash bits to locate. These slots are called the *first mapped slot*, the *second mapped slot*, ..., the *k_2th mapped slot* of the tag, respectively, as is illustrated in Fig.2.

After the reader maps all tags to slots, it determines in which mapped slot each tag will actually transmit: To begin with, the reader considers only the first mapped slots of the tags in T, identifies the singleton slots, and assigns these slots to the tags mapped to them. These slots are given an *index value* of 1, indicating that they are the *first* mapped slots of the assigned tags. Then the reader removes these slots/tags from further consideration. It repeats the above process for the second mapped slots of the remaining tags: identifying the singleton slots, assigning these slots to the tags mapped to them, and removing them from further consideration. These slots are given an index value of 2. This process is repeated all the way through the k_2th mapped slots. In the end, each tag is assigned to at most one slot to report its presence. After determining the slot-tag assignment, the reader produces an expected bitmap, which contains a bit one for each assigned slot (expected to be busy) and a bit zero for each unassigned slot (expected to be empty). Next the reader must inform the tags about the assignment for their reporting.

The reader initiates communication with a broadcast request carrying p_s and r. The request is followed by a sequence of $\frac{f_2}{l_2}$ sub-frames. The reader begins each sub-frame by transmitting a slot-index array, which contains one index value for each of the l_2 slots. The index value of an unassigned slot is zero. Each index is $\lceil \log_2(k_2 + 1) \rceil$ bits long, and the total length of the array is $l_2 \lceil \log_2(k_2 + 1) \rceil$. For example, if $k_2 = 3$ and $l_2 = 48$, the slot-index array takes 96 bits. The reader can broadcast such an array using a time slot of size for transmitting a 96-bit tag ID in 14.37ms.

All tags receive p_s and r. Sampling can be performed pseudo-randomly using the method in [5], [19], [20], [25], which is predictable by the reader. The tags know which sub-frames they are mapped to by computing $h(ID, r)$. Each tag waits for its sub-frame, and receives the corresponding slot-index array at the beginning of the sub-frame. It knows its first mapped slot through k_2th mapped slot from the hash bits in $h(ID, r)$. The tag examines its first mapped slot. If the corresponding slot index (from the received array) happens to be 1, the tag knows that it is assigned to this slot and should report its presence by transmitting in this slot. Otherwise, it examines its second mapped slot to see if the slot index is 2, and if so transmits in that slot. This process repeats until an assigned slot is identified, or otherwise the tag will not transmit.

The reader monitors the status of all slots in each sub-frame, converting every busy slot to bit 1 and every empty slot to bit 0. If it observes any expected busy slot to be empty, i.e., that a bit 1 in the pre-computed bitmap turns out to be 0, it announces a missing tag event. By increasing the sampling probability p_s or increasing the frame size f_2, we can make more tags to report in singleton slots, thus increasing the missing-tag detection probability in order to meet a pre-specified requirement, even under the presence of unfiltered unknown tags, as our analysis will show.

E. Cardinality Estimation

As we will discuss later, in order to obtain some parameters in our protocol, we need to first perform cardinality estimation on the entire set E. There are many solutions [5], [20], [24], [37] for fast cardinality estimation in RFID systems. In this paper, we adopt state-of-the-art SRC estimator proposed in [5] to estimate the number of tags in E. As is proven in [5], the overhead of SRC estimator is at most $\Theta(\frac{1}{\epsilon^2} + loglog(|E|))$, which is relatively small compared with the overhead of missing tag detection. After estimation of $|E|$, we can use it to obtain the cardinality of unknown tags as $|U| = |E| - n$, which will be used later in our optimization of execution time.

IV. Performance Analysis

In this section, we formally analyze the performance of our protocol and derive the execution time for each phase. As is discussed in [34], when the number of unknown tags is small, the interference from U can be neglected and phase one does not need to be executed. Therefore, we only consider the case when $|U|$ is large such that phase one is always needed.

A. Execution Time in Phase One

We first analyze the execution time in phase one.

From (1), we know that for a given false positive ratio p_1, the size of an original Bloom filter can be calculated as

$$f = -\frac{nk_1}{\ln(1 - p_1^{\frac{1}{k_1}})}. \tag{3}$$

After the construction of the original Bloom filter, we segment it based on the set bits. For a Bloom filter encoding n elements with k_1 hash functions, the number of ones in it is at most $n \cdot k_1$. Thus, on average the length L_1 of each segment will be

$$L_1 = \frac{f}{nk_1} = -\frac{1}{\ln(1 - p_1^{\frac{1}{k_1}})}. \tag{4}$$

As a result, the size of each segment in our compressed filter is $l_1 = \log_2(L_1 + 1)$ bits and the total number of segments $f_1 = n \cdot k_1$. Therefore, if we use T_{ID} to carry our compressed filter, the execution time t_1 in phase one can be calculated as:

$$t_1 = T_{ID} \cdot \frac{f_1}{\frac{96}{l_1}} = \frac{f_1 \cdot l_1}{96} \cdot T_{ID} = \frac{nk_1}{\ln 2} \cdot \frac{T_{ID}}{96} \cdot \ln(1 - \frac{1}{\ln(1 - p_1^{\frac{1}{k_1}})}). \tag{5}$$

Using this compressed filter, the majority of unknown tags in U will be filtered out. Let N^* be the tag set which consists of tags that remain active after phase one and its cardinality be $n^* = |N^*|$. We know that N^* consists of two parts:

1) tags in T that are not missing.
2) tags in U that are false positives.

For the first tag set, its cardinality n_1^* is the number of tags in T that are present, that is

$$n_1^* = n - m. \tag{6}$$

For the second tag set, since the false positive ratio of our compressed filter is p_1, the number n_2^* of unknown tags that can pass our Bloom filter can be calculated as:

$$n_2^* = |U| \cdot p_1. \tag{7}$$

Combining (6) and (7), we have

$$n^* = n_1^* + n_2^* = n - m + |U| \cdot p_1. \tag{8}$$

This number will be used later for our analysis of the execution time in phase two.

B. Execution Time in Phase Two

Now, we move forward to analyze the execution time in phase two. Since the false positive ratio p_1 is very small, we assume that $n_2^* \ll n$ and $n^* \approx n$.

First we need to derive the detection probability after one execution of phase two. The probability that any sampled tag is successfully assigned to a singleton slot after the ith mapping is given by

$$\begin{aligned} P_i =& (1 - P_{i-1}) \sum_{j=0}^{n} \binom{n}{j} (p_s \frac{l_2}{f_2})^j (1 - p_s \frac{l_2}{f_2})^{n-j}. \\ & (1 - \frac{1 - P_{i-1}}{l_2})^{j-1} \cdot (1 - \frac{jP_{i-1}}{l_2}) + P_{i-1}, 1 \le i \le k_2. \end{aligned} \tag{9}$$

where the first term is the probability that a tag which is not assigned to a singleton slot after the $(i-1)$ mappings is assigned to a singleton slot in the ith mapping, the second term is the probability that a tag is assigned to a singleton slot after $i-1$ mappings, and the rest parameters are already defined in previous sections. Therefore, after k_2 mappings, the detection probability in phase two is:

$$p_2 = 1 - (1 - p_s \times P_{k_2})^m. \tag{10}$$

where P_{k_2} is the probability that any sampled tag is assigned to a singleton slot after k_2 mappings. P_{k_2} can be computed recursively from (9) with $P_0 = 0$.

Recall in Section II-B that our system requires a detection probability of at least α for reporting a missing tag event. In order to satisfy the requirement of our system, we need to set

$$p_2 = \alpha. \tag{11}$$

Combining (10) and (11), we can obtain f_2. The execution time of phase two can be obtained as

$$t_2 = f_2 \cdot T_{short} + \frac{f_2}{l_2} \cdot T_{ID}. \tag{12}$$

P_{fp}^{theo}	0.005	0.001	0.0005	0.0001
optimal size	11027	14377	15820	19170
compressed size	10879	13174	13912	16511
actual P_{fp}^{opt}	0.0050	0.0010	0.00053	0.00010
actual P_{fp}^{cmp}	0.0050	0.0010	0.00050	0.00008

TABLE I: Comparison between the optimal Bloom filter and our compressed filter.

V. Simulation Results

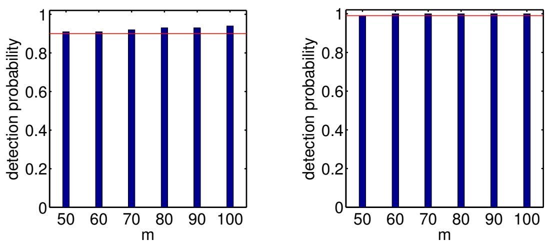
We conduct extensive simulations to evaluate the performance of our protocol. There is limited work on missing tag detection with presence of unknown tags. Only RUN [25] and BMTD [34] can detect a missing tag event with the required probability in our scenario, while most of the existing work cannot handle the interference of unknown tags. Thus, we compare our protocol with RUN and BMTD. For each set of experiments, we repeat 100 times under the same simulation settings. The parameters of RUN and BMTD are set based on [25] and [34], respectively.

A. Efficiency of Compressed Filter

The first set of experiments study the efficiency of our compressed filter. In our simulation, we set the size of set T as 1000 and set the false positive ratios as 0.005, 0.001, 0.0005 and 0.0001 respectively. We use 10000 elements to test the actual false positive ratio of the filters.

Table I shows the performance of the optimal Bloom filter and our compressed filter. In this table, the first row shows the theoretical false positive ratio that we set in the simulations. The second row lists the sizes of the optimal Bloom filter that are calculated based on (2). These results agree with our analysis that the size of the Bloom filter increases when P_{fp} becomes lower. The third row shows the sizes of our compressed filter using our compression algorithm, which are much smaller than the sizes of the optimal Bloom filter in the second row under the same false positive ratio requirement. This proves that our compressed filter is more efficient than the optimal Bloom filter. Specifically, when the false positive ratio is required to be 0.0001, our compressed filter saves 13.8% space compared with the optimal Bloom filter. The last two rows show the actual false positive ratio of the optimal Bloom filter and our compressed filter, respectively. We can observe that the false positive ratios of both filters are close to the theoretical one, while the false positive ratio of our compressed filter can be lower than that of the optimal one. These results show that our compressed filter can achieve an even smaller false positive ratio, which can filter out more unknown tags, with a smaller size than the optimal Bloom filter. All these results demonstrate the effectiveness of our design.

However, this does not mean that the design of original Bloom filter and optimal Bloom filter is not compact. We want to point out that for the problem of missing-tag detection with presence of unknown tags, when the required false positive ratio is very small, we can compress the Bloom filter to save more space. While for other problems, the required false positive ratio may not be as small, the design of original Bloom filter is still capable of space-efficiently representing the data set.



(a) α = 0.9. (b) α = 0.99.

Fig. 3: Reliability of our protocol.

B. Accuracy

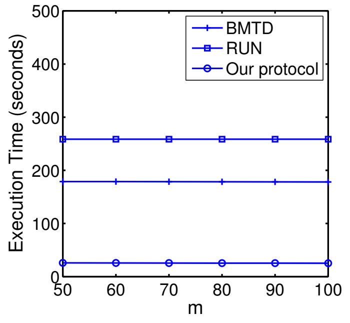
The second set of experiments investigate the accuracy of our protocol. We want to verify that our protocol can detect an missing tag even with the required detection probability α, which is satisfied by RUN and BMTD.

In our simulations, we set $n = 10000$, $M = 1$ and $|U| = 50000$. We vary the number m of missing tags in our system from 50 to 100 at a step size of 10 and set the detection probability α as 0.9 and 0.99 respectively. Other parameters of our protocol are set as is described in Section IV.

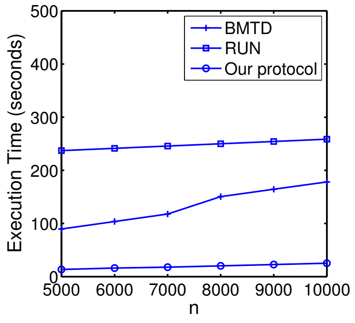
Fig. 3 shows the actual detection probability of our protocol with different detection probability. The left plot shows the actual detection probability when $\alpha = 0.9$ and m varies from 50 to 100, while the right one corresponds to the accuracy requirement of $\alpha = 0.99$. In each plot, the x coordinate is the number m of missing tags and the y coordinate is actual detection probability. The red line is the required detection probability and each bar represents the average actual detection probability of 100 runs under the given setting. We can observe that the actual detection probability is always higher than the required one. Besides, as the number of missing tags increases, the detection probability becomes higher, which is expected since it is easier to detect a given number of tags when more tags are actually missing.

C. Execution Time

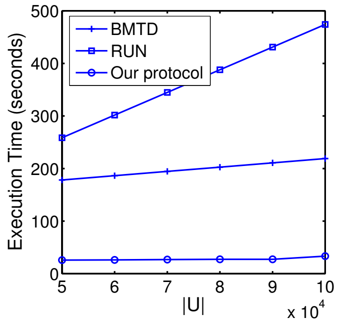
We evaluate and compare the execution time of our protocol with RUN and BMTD through simulations. In our simulation, we set $M = 1$, and let $\alpha = 0.9$ and 0.99, respectively. We vary m, n and $|U|$ to investigate their impact on the time-efficiency of these three missing-tag detection protocols. We use real time (in seconds) for evaluation. Recall that in Section III-C, we configure the EPC standard to transmit two different types of slots: T_{short} and T_{ID}. For RUN, we use T_{short} for tags to transmit their responses and the total execution time is $f_{RUN} \times T_{short}$, where f_{RUN} is the optimal frame size that is obtained from [25]. For BMTD, we use T_{ID} for the reader to broadcast the Bloom filter in 96-bit segments and use T_{short} for tags to transmit their responses. The total execution time



(a) Impact of m.

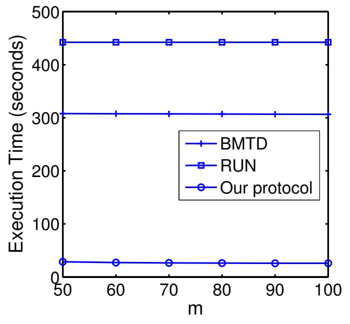


(b) Impact of n.

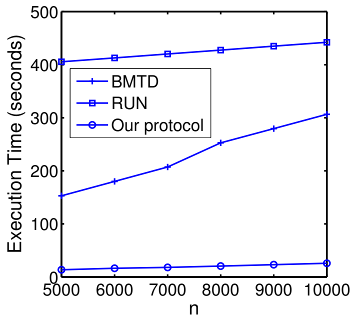


(c) Impact of $|U|$.

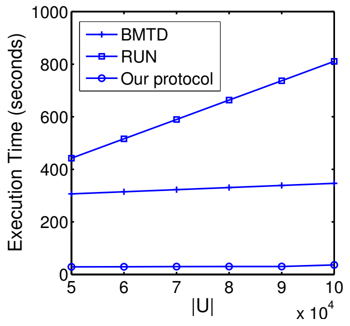
Fig. 4: Execution time of different protocols when $\alpha = 0.9$.



(a) Impact of m.



(b) Impact of n.



(c) Impact of $|U|$.

Fig. 5: Execution time of different protocols when $\alpha = 0.99$.

is $f_{BMTD1} \times \frac{T_{ID}}{96} + f_{BMTD2} \times T_{short}$, where f_{BMTD1} and f_{BMTD2} are the optimal frames size of phase one and phase two in BMTD that are obtained from [34]. The parameters for our protocol are set based on Section IV.

1) Impact of m: The third set of experiments study the impact of the number of missing tags. In our simulations, we set $n = 10000$, $|U| = 50000$ and vary m from 50 to 100 at a step size of 10. The experiments are conducted under accuracy requirement of both $\alpha = 0.9$ and 0.99.

Fig. 4a and Fig. 5a show the results of our simulations under different accuracy requirements. In each plot, the x coordinate is the number of missing tags and the y coordinate is the overall execution time (in seconds). It is expected that the execution times of these three protocols increase as the detection probability increases. Besides, we can observe that the execution times of these three protocols decrease (but slowly) with the increase of m, which is expected since for a given threshold, the more tags are missing, the faster we can detect a missing-tag event. In addition, our protocol takes less time than BMTD and RUN for detection. The execution time is reduced by 83% compared with the state-of-the-art (BMTD), when $\alpha = 0.9$. When $\alpha = 0.99$, our protocol is even better compared with BMTD and RUN. Specifically, when $\alpha = 0.9, m = 50$, the execution times of our protocol, BMTD and RUN are $25.82, 178.90$ and 258.68 *seconds*, respectively. This comes from the fact that RUN does not filter out any unknown tags in its detection, thus will waste much time on these tags. BMTD tries to filter out these unknown tags, but the Bloom filter it uses is less efficient than our design. As a result, our protocol outperforms RUN and BMTD in missing-tag detection with presence of unknown tags.

2) Impact of n: The fourth set of experiments study the impact of the number of tags in T. In our simulations, we set $m = 100$, $|U| = 50000$ and vary n from 5000 to 10000 at a step size of 1000. The experiments are conduct under accuracy requirement of both $\alpha = 0.9$ and 0.99.

Fig. 4b and Fig. 5b show the results of our simulations under different accuracy requirements. Similarly, the execution times increase as the detection probability increases. Besides, the execution times of all three protocols increase with respect to n, but our protocol still outperforms BMTD and RUN and consumes much less time (6 times shorter than BMTD and 15 times short than RUN when $\alpha = 0.9$). The raise of execution times is expected since a larger frame needs to be allocated to maintain the detection accuracy, when the number of known tags increases. In specific, when $\alpha = 0.9, n = 5000$, the execution times of our protocol, BMTD and RUN are $13.44, 89.55$ and 237.13 *seconds*, respectively. Overall, our protocol outperforms RUN and BMTD in missing-tag detection with presence of unknown tags under both accuracy requirements.

3) Impact of $|U|$: The fifth set of experiments study the impact of the number of unknown tags. In our simulations, we set $n = 10000$, $m = 100$ and vary $|U|$ from 50000 to 100000 at a step size of 10000. The experiments are conduct under accuracy requirement of both $\alpha = 0.9$ and 0.99.

Fig. 4c and Fig. 5c show the results of our simulations under different accuracy requirements. We can observe that as the number of unknown tags increases, the execution time of RUN increases drastically, while the execution time of our protocol and BMTD increases slowly. This agrees with our analysis that the performance of RUN drops a lot when dealing with large tag sets since it does not filter out unknown tags. While both our protocol and BMTD filter out the inference of unknown tags, the design of our compressed filter is more efficient as is analyzed previously. All these results above demonstrate that our protocol can more time-efficiently perform reliable missing tag detection than RUN and BMTD with large unknown tag sets.

VI. Conclusion

In this paper, we propose a new protocol that performs reliable and efficient missing-tag detection with presence of unknown tags. We design a compressed filter which achieves a comparable false positive ratio to Bloom filter with a smaller size and propose new collision-reduction methods to increase our efficiency. Extensive simulations show that our protocol outperforms existing works and that when comparing with the best prior work, more than 83% execution time is saved in detecting a missing-tag event with presence of unknown tags, while the accuracy requirement is still satisfied.

VII. Acknowledgements

This work was in part supported by the National Science Foundation under grants CNS-1409797 and CNS-1718708.

References

[1] Data Compression. *https://en.wikipedia.org/wiki/Data_compression*.

[2] EPC Radio-Frequency Identity Protocols Class-1 Gen-2 UHF RFID Protocol for Communications at 860MHz-960MHz, EPCglobal. *http://www.epcglobalinc.org/uhfclg2*.

[3] National Retail Federation, National retail security survey. *https://nrf.com/resources/retail-library/national-retail-security-survey-2015*.

[4] M. Buettner and D. Wetherall. A software radio-based UHF RFID reader for PHY/MAC experimentation. *Proc. of IEEE RFID*, 2011.

[5] B. Chen, Z. Zhou, and H. Yu. Understanding RFID Counting Protocols. *Proc. of Mobicom*, 2013.

[6] M. Chen, S. Chen, Y. Zhou, and Y. Zhang. Identifying State-free Networked Tags. *IEEE/ACM Transactions on Networking*, 2017.

[7] M. Chen, W. Luo, Z. Mo, S. Chen, and Y. Fang. An Efficient Tag Search Protocol in Large-Scale RFID Systems. *Proc. of IEEE INFOCOM*, April 2013.

[8] D. Ciuonzo, A. D. Maio, and P. S. Rossi. A systematic framework for composite hypothesis testing of independent Bernoulli trials. *IEEE Signal Processing Letters*, 22(9):1249 – 1253, 2015.

[9] D. Ciuonzo and P. S. Rossi. Decision fusion with unknown sensor detection probability. *IEEE Signal Processing Letters*, 21(2):208 – 212, 2014.

[10] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *Proc. of SIGCOMM*, 1998.

[11] J. Han, Q. Chen, P. Yang, D. Ma, Z. Jiang, W. Xi, and J. Zhao. GenePrint: Generic and Accurate Physical-Layer Identification for UHF RFID Tags. *IEEE/ACM Transactions on Networking*, 24(2):846 – 858, 2016.

[12] Y. Hou, J. Ou, Y. Zheng, and M. Li. PLACE: Physical Layer Cardinality Estimation for Large-Scale RFID Systems. *IEEE/ACM Transactions on Networking*, 2010.

[13] T. Li, S. Chen, and Y. Ling. Identifying the Missing Tags in a Large RFID System. *Proc. of ACM Mobihoc*, 2010.

[14] J. Liu, Y. Zhang, M. Chen, S. Chen, and L. Chen. Collision-resistant Communication Model for Stateless Networked Tags, poster paper. *Proc. of IEEE ICNP*, 2016.

[15] X. Liu, K. Li, S. Guo, A. X. Liu, P. Li, K. Wang, and J. Wu. Top-K Queries for Categorized RFID Systems . *IEEE/ACM Transactions on Networking*, 2016.

[16] X. Liu, K. Li, G. Min, Y. Shen, A. X. Liu, and W. Qu. Completely pinpointing the missing RFID tags in a time-efficient way. *IEEE Transaction on Computers*, 64(1):87 – 96, 2015.

[17] X. Liu, B. Xiao, K. Li, J. Wu, A. X. Liu, H. Qi, and X. Xie. RFID Cardinality Estimation with Blocker Tags. *Proc. of IEEE INFOCOM*, 2015.

[18] W. Luo, S. Chen, T. Li, and S. Chen. Probabilistic Missing-tag Detection and Energy-Time Tradeoff in Large-scale RFID Systems. *Proc. of ACM Mobihoc*, 2012.

[19] W. Luo, Y. Qiao, S. Chen, and T. Li. Missing-Tag Detection and Energy-Time Tradeoff in Large-Scale RFID Systems with Unreliable Channels. *IEEE/ACM Transactions on Networking*, 2014.

[20] W. Luo, S. Wu, S. Chen, and M. Yang. Energy Efficient Algorithms for the RFID Estimatino Problem. *Proc. of IEEE INFOCOM*, 2010.

[21] M. Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM Transaction on Networking*, 10(5):604 – 612, 2002.

[22] J. Ou, M. Li, and Y. Zheng. Come and Be Served: Parallel Decoding for COTS RFID Tags. *Proc. of ACM MobiCom*, 2015.

[23] S. Qi, Y. Zhang, M. Li, Y. Liu, and J. Qiu. Scalable Data Access Control in RFID-Enabled Supply Chain. *Proc. of IEEE ICNP*, 2014.

[24] M. Shahzad and A. X. Liu. Every Bit Counts - Fast and Scalable RFID Estimation. *Proc. of ACM Mobicom*, 2012.

[25] M. Shahzad and A. X. Liu. Expecting the Unexpected: Fast and Reliable Detection of Missing RFID Tags in the Wild. *Proc. of IEEE INFOCOM*, 2015.

[26] M. Shahzad and A. X. Liu. Fast and Accurate Tracking of Population Dynamics in RFID Systems. *Proc. of IEEE ICDCS*, 2017.

[27] L. Shangguan, Z. Zhou, X. Zheng, L. Yang, Y. Liu, and J. Han. ShopMiner: Mining Customer Shopping Behavior in Physical Clothing Stores with Passive RFIDs. *Proc. of IEEE INFOCOM*, 2016.

[28] A. D. Smith, A. A. Smith, and D. L. Baker. Inventory management shrinkage and employee anti-theft approaches. *International Journal of Electronic Finance*, 5(3):209 – 234, 2011.

[29] C. C. Tan, B. Sheng, and Q. Li. How to monitor for missing RFID tags. *Proc. of IEEE ICDCS*, 2008.

[30] 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. *Proc. of IEEE SECON*, 2017.

[31] Q. Xiao, S. Chen, and M. Chen. Joint Property Estimation for Multiple RFID Tag Sets using Snapshots of Variable Lengths. *Proc. of ACM Mobihoc*, 2016.

[32] L. Yang, Y. Chen, X. Li, C. Xiao, M. Li, and Y. Liu. Tagoram: Real-Time Tracking of Mobile RFID Tags to High Precision Using COTS Devices. *Proc. of ACM MobiCom*, 2014.

[33] L. Yang, Q. Lin, X. Li, T. Liu, and Y. Liu. See Through Walls with COTS RFID System. *Proc. of ACM MobiCom*, 2015.

[34] J. Yu, L. Chen, R. Zhang, and K. Wang. Finding needles in a haystack: Missing tag detection in large rfid systems. *IEEE Trans. on Communication*, 65(5):2036 – 2047, 2017.

[35] R. Zhang, Y. Liu, Y. Zhang, and J. Sun. Fast Identification of the missing tags in a large RFID system. *Proc. of IEEE SECON*, 2011.

[36] Y. Zhang, S. Chen, Y. Zhou, and Y. Fang. Anonymous Temporal-Spatial Joint Estimation at Category Level over Multiple Tag Sets. *Proc. of IEEE INFOCOM*, 2018.

[37] Y. Zheng and M. Li. Fast Cardinality Estimation for Large-scale RFID Systems. *Proc. of IEEE INFOCOM*, 2013.

[38] Y. Zheng, M. Li, and C. Qian. PET: Probabilistic Estimating Tree for Large-Scale RFID Estimation. *Proc. of IEEE ICDCS*, 2011.