Missing-Tag Detection With Unknown Tags

Youlin Zhang, *Member, IEEE*, Shigang Chen, *Fellow, IEEE*, You Zhou, *Member, IEEE*, and Yuguang Fang, *Fellow, IEEE, Member, ACM*

***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 new missing tag detection protocol based on compressed filters, which not only reduce the filter size for better time-efficiency but also help dampen the interference of unknown tags for high missing-tag detection accuracy. To further improve the performance, we propose to use a combination of sampling and multi-hashing for tags to report their presence, greatly reducing collisions and thus improving the detection probability. We reconfigure the standard ID collection protocol to support bitmap collection required by missing-tag detection. 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. We also evaluate the performance of our missing-tag detection protocol under unreliable channel.**

***Index Terms*— Radio Frequency Identification (RFID) tags, wireless application protocol.**

I. Introduction

In recent years, RFID technologies have been proliferating, with numerous applications that have been developed including supply chain management [1]–[18], object tracking [19]–[21], theft prevention [22]–[28] and so on [29]–[33]. 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.

Manuscript received September 6, 2018; revised May 7, 2019 and December 21, 2019; accepted March 6, 2020; approved by IEEE/ACM Transactions on Networking Editor M. Li. This work was supported in part by the National Science Foundation under Grant CNS-1409797, Grant CNS-1718708, and Grant CNS-1719222 and in part by a grant from Cyber Florida. *(Corresponding author: Youlin Zhang.)*

Youlin Zhang, Shigang Chen, and You Zhou are with the Department of the Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: youlin@cise.ufl.edu).

Yuguang Fang is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA.

Digital Object Identifier 10.1109/TNET.2020.2984706

One of the most popular applications using RFID tags is to automatically detect missing products in a storage place. According to [34], [35], 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 [23]–[25] and probabilistic detection [22], [26]–[28]. 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 baggage 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 [22], [26], [36] cannot handle unknown tags. It is shown in [28] that the presence of unknown tags will introduce false positives and compromise the detection accuracy. The most related work is [27], [28], which can detect missing-tag events with unknown tags. The efficiency of [28] 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 [27] 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 [1]–[8], 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. 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. We provide optimal parameter setting that can tolerate the noise from unknown tags to achieve a pre-specified detection accuracy.

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. Combined with a new segmentation mechanism, our compressed filter allows tags to perform membership check without decompressing the filter and its size is much smaller than the original Bloom filter used by the prior art.

Third, we formally analyze the performance of our protocol. Through analysis, we investigate the impact of different parameters on the execution time of our protocol and optimize the parameters to minimize execution time.

Fourth, we reconfigure the standard ID collection protocol to support bitmap collection required by missing-tag detection.

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. We also evaluate the performance of our missing-tag detection protocol under unreliable channel.

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 tags 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 channel state 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.

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. Related Work

There are two types of missing tag detection protocols: deterministic detection and probabilistic detection. In this section, we will briefly review these related missing tag detection protocols and point out the issues existed in them.

The objective of deterministic detection is to exactly identify which tags are missing and to report the IDs of missing tags. It can produce strong results but requires huge overhead. A straightforward approach is to broadcast each tag ID and identify tags in T one by one. If some tags are missing, there will be no respond from those tags and their status are identified. And the time cost of this approach is $96 \cdot n$, where 96 is the length of a tag ID and it is very time-consuming. Li *et al.* [23] make an improvement over this approach by resorting to the properties of hash functions. And they propose a series of protocols including TPP, TPP/CSTR and IIP to reduce the hash collisions in tag identification and achieve missing tag detection without transmitting tag IDs in large RFID systems. Zhang *et al.* [24] use bitmaps to store the status of all tags and identify the IDs of missing tags by comparing the pre-computed bitmaps. Moreover, Liu *et al.* [25] improve the performance of previous work by reconciling 2-collision and 3-collision slots and filtering the empty slots. However, all these protocols are designed for deterministic detection, which are time-consuming and infeasible for probabilistic detection. Besides, they all work in restricted environment settings, which requires full-awareness of all tag IDs and can not be applied to more open scenarios when unknown tags are present [28].

The objective of probabilistic detection is to detect a missing tag event with a predefined probability, which is more related to our work. Tan *el al.* [22] propose a bitmap-based missing tag detection protocol called TRP. TRP compares the bitmap which encodes the responses of present tags with the pre-computed bitmap of all known tags and reports a missing tag event when some bits mismatch. Luo *et al.* [26] make an improvement over [22] and propose a multi-seed missing tag detection protocol called MSMD, which employs multi-seed mapping to reduce the probability of collisions in the communication. However, both of them require full knowledge of tag IDs. When applied to a more open scenario where unknown tags are present, their accuracy drops drastically and can not satisfy the predefined requirement [28]. RUN [28] and BMTD [27] take the first step to solve the problem of missing tag detection with presence of unknown tags. RUN uses bitmaps to estimate the interference from unknown tags. Theoretical analysis on the performance of RUN is provided to optimize the parameters in the system. But the efficiency of RUN will drop in the presence of a large number of unknown tags [27], since it does not filter out unknown tags. BMTD leverages Bloom filter to reduce the interference of unknown tags and achieves a better performance than RUN. However, its time efficiency will degrade when the size of the tag set grows larger.

All these problems existed in the prior work drive us to explore innovative ways for missing tag detection with presence of unknown tags. We propose a compressed filter based protocol, which can reliably and time-efficiently detect a missing tag event when unknown tags are present.

IV. 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 [27] 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 [27] which uses a design based on Bloom filters [37], a compact data structure that is used to encode a set $T = \{t_1, t_2, \ldots, t_n\}$ 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 $h_1, h_2, \ldots, h_k$ 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 [37]:

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

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.

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

An optimal filter has 50% zeros, and therefore its false positive ratio is $(\frac{1}{2})^k$. 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. When the size f is restricted, 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, we 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. We have to use non-optimal filters.

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. One can compress a Bloom filter to minimum size by using optimal compression algorithms in the literature [38]. However, if we do that in RFID systems, there arise two problems. First, tags are resource-limited and cannot perform generic compression and decompression algorithms such as Huffman compression and LZ compression [39]. 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

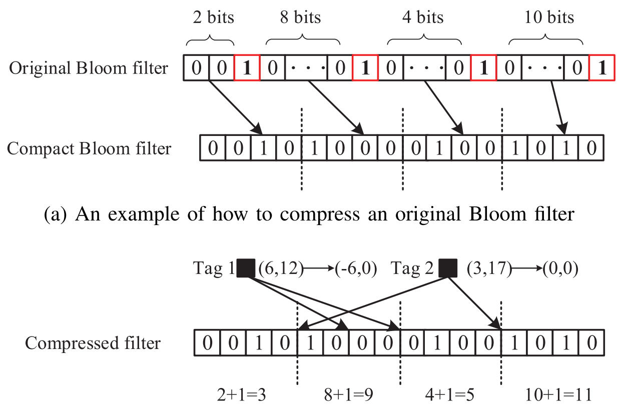
To reduce the interference of unknown tags, the reader uses a compressed filter with a small pre-set false positive ratio p_1 such that the majority of unknown tags are filtered and do not participate in Phase two. Let k_1 be the number of hash functions used in the filter of Phase one. We design a compressed 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. As we will discuss later in our evaluation, the compressed filter will have a higher compression ratio in our application scenario when k_1 and p_1 are very small.

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 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 the same time slots 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_1 bits, the tag loads the hash position of the bit in the filter into a segment counter.



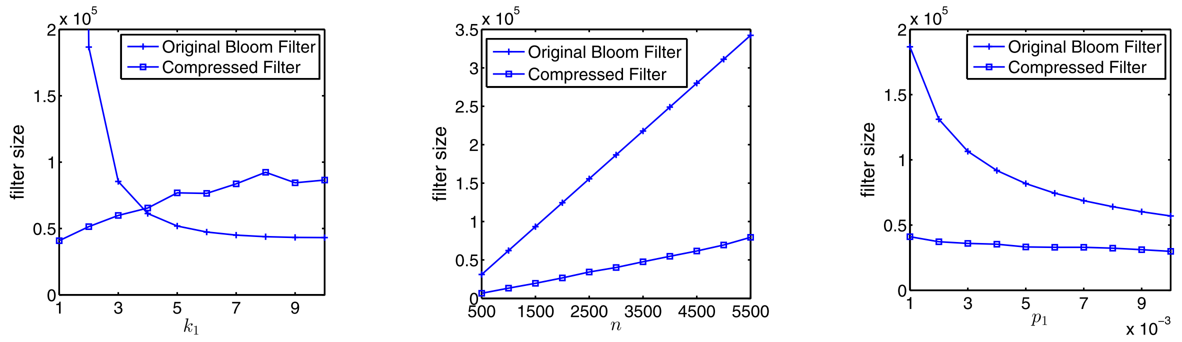
(b) An example of how to use a compressed filter for membership check

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

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.

We perform experiments to compare the size of our compressed filter with that of the original Bloom filter. The results are shown in Fig. 2, consisting of three plots that compare the two filters' sizes with respect to the number k_1 of hash functions, the number n of encoded tags, and the false positive ratio requirement p_1, respectively. In each plot, the x-axis



(a) number k_1 of hashes (b) number n of encoded tags (c) false positive ratio p_1.

Fig. 2. Comparison between the compressed filter and the original Bloom filter.

represents a parameter setting (k_1, n or p_1), while the y-axis shows the filter size. Fig. 2a compares the two filters with respect to k_1, where $n = 3000$ and $p_1 = 0.001$. Recall that the motivation behind the design of our compressed filter is to work with a small number of hash functions, considering the limited tag resources. The size of the compressed filter is just 40838 when $k_1 = 1$. In comparison, the size of the Bloom filter is 2998500 when $k_1 = 1$, 85420 when $k_1 = 3$, and 43130 when k_1 takes its optimal value of 10. An interesting observation is that the size of the compressed filter first increases as k_1 increases, then the size decreases. The reason is that a larger number of hashes will set more bits in the filter to ones and thus create more segments in the compressed filter, but the average segment size may decrease. Overall, the size of our compressed filter reaches the smallest when $k_1 = 1$ as we find out in our experiments. Fig. 2b compares the two filters with respect to n, where k_1 is set to 1 for the compressed filter and 3 for the Bloom filter, and $p_1 = 0.001$. The sizes of both filters increase as n increases, while our compressed filter is always smaller than the Bloom filter, even with a fewer number of hashes. Fig. 2c compares the two filters with respect to p_1, where $n = 3000$ and k_1 is set to 1 for the compressed filter and 3 for the Bloom filter. The sizes of both filters increase as p_1 decreases, while our compressed filter is always smaller than the Bloom filter, even with a fewer number of hashes.

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 [27], 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 [5]–[13]. We have implemented such a design that conforms to the EPC C1G2 standard [40] following [41]. Details of the implementation is provided in Section VI, and a brief summary is given as follows: In the standard C1G2 protocol for tag ID collection, 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 → RN16 → ACK → 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 denote the time of such a slot as T_{ID}, during which 96 bits (i.e., a tag ID) are exchanged between a reader and a tag. We have reconfigured the above protocol to deliver one bit information in each slot, indicating the presence of a tag (without transmitting its ID). In the modified protocol, the transmissions of ACK and tag ID are skipped. After RN16, the reader immediately sends QueryRep 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 [27], 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 → RN16, and its time is denoted as T_{short}. T_{short} translates into 2.69ms based on the settings in [41]. In comparison, the original slot structure for tag ID, i.e., T_{ID}, takes 14.37ms.

BMTD [27] is inefficient because it requires many bits (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 in a slotted time frame initiated by the reader. Suppose we assign each tag to a single slot by hashing the tag ID. The slots can be classified into three types: singleton slots, collision slots, and empty slots, which have a single assigned tag, multiple assigned 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 samples the tags in T and initiates a slotted time frame, where each sampled tag will transmit in one of the slots. The reader will convert the time frame to a bitmap, each slot for one bit, which is one if the slot is busy or zero if the slot is empty.

Let the number of slots in the time frame be f_2, which is also the number of bits in the bitmap. The time frame 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 k_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.3. (We will discuss how to set the optimal values of the system parameters, including f_2, l_2 and k_2, in the next section.)

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 pre-computes 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 a sampling probability p_s and a random seed 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.

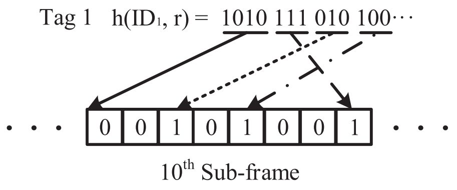


Fig. 3. An illustration of multi-hashing, where $k_2 = 3$ and $l_2 = 8$. The first four bits of $h(ID_1, r)$ are used to decide in which sub-frame (the 10th in this example) the tag will transmit. The first three bits after those four bits are used to decide the first mapped slot of this tag in this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

All tags receive p_s and r. Sampling can be performed pseudo-randomly using the method in [1], [4], [28], [36], 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 [1], [3], [4], [7], [8] for fast cardinality estimation in RFID systems. In this paper, we adopt state-of-the-art SRC estimator proposed in [4] to estimate the number of tags in E. The SRC estimator takes a deeper insight into existing RFID counting protocols and designs a more efficient algorithm based on existing literature [3], [42], [43]. The protocol consists of two phases: a rough estimation phase, which uses a small overhead of $\Theta(loglog(|E|))$ to get a rough cardinality estimation, and an accurate estimation phase, which uses $\frac{65}{(1-0.04^{\epsilon})^2}$ (ϵ is the relative estimation error) slots to obtain an accurate result. For example, when $\epsilon = 0.1$ as BMTD [27] sets, in our simulations the number of slots calculated from [4] is around 1000, which is relative small compared with the slots needed by RUN, BMTD and our protocol (more than 30000). 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.

F. Hash Function

There are many efficient implementations of hash functions in the literature. To lower the complexity of the tag circuit, we implement our hash function based on the scheme in [23], which uses a ring of pre-stored random bits: Before a tag is deployed, the ID of a tags is used as seed for a random number generator to produce a string of pseudo-random bits. These bits are stored in the tag and form a logical ring. After deployment of a tag, it generates a hash value $h(id, r)$ by returning a certain number of bits after the rth bit in the ring, where id is the tag ID and r is a hash seed. This hash value can be predicted by an RFID reader given the tag ID and the seed r.

This design of hash function is easy to implement in hardware and thus suitable for tags. But it can only produce a limited number of different hash values for each tag, depending on the size of the ring. It is not suitable for protocols that require each tag to produce a large number of different hash values. However, as we will see later, in our protocol, we only require each tag to produce a few hash values for use, in specific, only 1 hash value in Phase one and 3 in Phase two, which makes our implementation work well.

V. Performance Analysis and Parameter Optimization

In this section, we formally analyze the performance of our protocol and optimize the parameters in our protocol such that the execution time is minimized while the detection probability is satisfied. As is discussed in [27], 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 bits of ones. For a Bloom filter encoding n elements with k_1 hash functions, the number of ones in it is at most $n \cdot k_1$, which is an upper bound for the number f_1 of segments of consecutive zeros in the filter. Thus, the average segment length is lower-bounded by

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

Recall that f_1 denotes the size of the compressed filter in Phase one and l_1 denotes the number of bits used to encode each segment. The value of l_1 is lower-bounded by $\log_2(L_1 + 1)$ bits.

Suppose we use the same time slots for transmitting 96-bit tag IDs to transmit the filter. Phase one will take $\frac{f_1}{\frac{96}{l_1}}$ slots, each transmitting $\frac{96}{l_1}$ segments in time T_{ID}, which is defined in Section IV-C. Therefore, the execution time t_1 in Phase one is

$$t_1 = T_{ID} \cdot \frac{f_1}{\frac{96}{l_1}}. \tag{5}$$

With the compressed filter, we want to filter the majority of unknown tags in U. 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

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

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 number of bits in the bitmap of Phase two, each taking a short time slot

T_{short} to deliver, as explained in Section IV-C. The execution time of Phase two can be obtained as

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

where the first term on the right is the time for tags to report their presence using the bitmap protocol and the second term is the time for the reader to transmit the slot-index arrays.

C. Overall Execution Time

Finally, we can obtain the overall execution time of our protocol based on the analysis in Section V-A and Section V-B as follows:

$$\begin{aligned} t &= t_1 + t_2 \\ &= \frac{f_1 \cdot l_1}{96} \cdot T_{ID} + f_2 \cdot T_{short} + \frac{f_2}{l_2} \cdot T_{ID}. \end{aligned} \tag{13}$$

In order to improve the performance of our protocol, we need to minimize t such that a missing tag event can be detected and reported in the shortest time. So in this section, we study the impact of different parameters on our execution time t. For a given RFID system, some parameters in (13) such as $m, n, |U|$ and α are set by users. Thus, to minimize the time cost, we mainly investigate the impact of parameters including k_1, p_1, k_2 and p_s.

We want to first point out that the increase of the number k_2 of mappings in Phase two will increase the detection probability and reduce detection time. However, more mappings will increase the computation overhead for tags and will also require more on-chip memory to store the computation results. Therefore, to simply the operation of tags, we set $k_2 = 3$ in both our analysis and experiments when using multi-hashing in Phase two.[1]

1) Impact of k_1: First we study how the number k_1 of hash functions used in Phase one will affect the execution time of our protocol. Since k_1 does not affect the execution time t_2 in Phase two, we only focus on t_1 in our analysis.

We take (4) to (5) and take the first derivative of t_1 with respect to k. Then we have

$$\begin{aligned} t_1'(k_1) = [&\ln(1 - \frac{1}{\ln(1-p_1^{\frac{1}{k_1}})}) \\ &+ \frac{p_1^{\frac{1}{k_1}} \ln p_1}{k_1(1-p_1^{\frac{1}{k_1}})(1-\frac{1}{\ln(1-p_1^{\frac{1}{k_1}})})\ln^2(1-p_1^{\frac{1}{k_1}})}] \cdot \frac{nT_{ID}}{96}. \end{aligned} \tag{14}$$

If we take the second derivative, we can find that $t_1''(k_1) > 0, \forall k_1 \in [1, \infty)$ and k_1 is an integer. Thus, $t_1'(k_1)$ is increasing with respect to k_1 in its domain. Besides, we can obtain from (14) that $t_1'(1) > 0$ for $\forall p_1 \in (0, 1)$. As a result, $t_1'(k_1) > 0, \forall k_1 \in [1, \infty)$ and $t_1(k_1)$ is increasing with respect to k_1 in $[1, +\infty)$. Therefore, the execution time t_1 (and t) is minimized when $k_1 = 1$. In the rest of our paper, we set $k_1 = 1$ to simplify our analysis.

Fig. 4a shows the relationship between execution time t_1 and k_1 when $n = 10000, m = 100, |U| = 50000$ and $\alpha = 0.99$. In this figure, the x coordinate is the number of hash functions and the y coordinate is the execution time (in seconds) for Phase one. We can observe that execution time is increasing as the number of hash functions increases and will reach its minimum when $k_1 = 1$.

[1]One may ask why we do not use $k_2 = 2$. The reason is that we will always need 2 bits for each slot index when $k_2 = 2$ or 3. In this condition, we should choose $k_2 = 3$ for better performance.

2) Impact of p_1: It is hard to obtain an explicit expression of t with respect to the false positive ratio p_1 from (13). However, we want p_1 to be smaller such that the missing tag detection is more reliable. In Section V-B, we assume $n_2^* \ll n$, that is $p_1 \cdot |U| \ll n$. Therefore, we have

$$\frac{p_1 \cdot |U|}{n} \ll 1. \tag{15}$$

In practice, we set $\frac{p_1 \cdot |U|}{n} = \delta$, where $\delta \ll 1$ such that (15) will hold. For example, we set $\delta = 10^{-3}$, when $n = 10000, |U| = 50000$. In this case, there will only be approximately 10 tags from U that participate in Phase two, which can be neglected as Section V-B does.

3) Impact of p_s: In this section, we study how the sample probability p_s influences the execution time of our protocol. Since p_s does not affect the execution time t_1 in Phase one, we only focus on f_2/t_2 in our analysis.

Combining (10) and (11), we can observe that for each different sampling probability p_s, we can compute an optimal frame size f_2^* that satisfied the detection probability using the bi-section algorithm. If we consider the optimal value f_2^* as a function of p_s, denoted as $f_2^*(p_s)$, $f_2^*(p_s)$ can be optimized as:

$$f_2^*(p_s) = min\{f_2 | p_2 \geq \alpha\}, \text{ s.t. } f_2 \leq F, f_2 \in I^+. \tag{16}$$

where F is the upper bound of acceptable frame size and I^+ is the set of positive integers. $f_2^*(p_s)$ can be computed based on bi-section search using Algorithm 1.

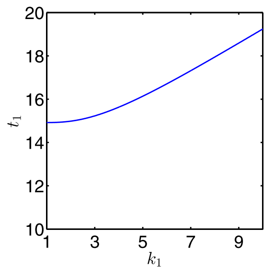
Algorithm 1 Search for $f_2^*(p_s)$

Input: m, n_*, α, k_2, p_s

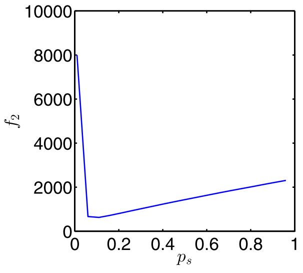
Result: the minimal frame size under sampling probability p_s

```
1:  if $p_2 < \alpha$ then
2:      exit;
3:  end if
4:  $f_s = 1,\ f_e = F$;
5:  while $f_e - f_s > 1$ do
6:      $f_m = \lceil \frac{f_s + f_e}{2} \rceil$;
7:      if $p_2 < \alpha$ then
8:          $f_s = f_m$;
9:      else
10:         $f_e = f_m$;
11:     end if
12: end while
13: return $f_e$
```

Fig. 4b shows the tradeoff between f_2 and p_s when $n = 10000, m = 100, |U| = 50000$ and $\alpha = 0.99$. In this figure, the x coordinate is the sampling probability p_s and the y coordinate is the frame size for Phase two. We can observe that there is a tradeoff between f_2 and p_s. An optimal p_s can be determined by searching p_s in the range of $(0, 1]$ such that $f_2^*(p_s)$ is minimized.



(a) Impact of k_1.



(b) Impact of p_s

Fig. 4. Impact of different parameters on execution time.

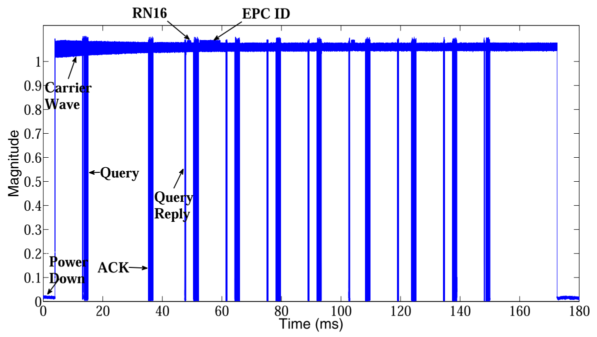


Fig. 5. Communication between a reader and tags for tag ID collection.

VI. Implementation of Bitmap Protocol

The proposed solution of missing tag detection with unknown tags requires each tag to perform some simple calculations for membership check and report their presence in the form of a bitmap. In specific, tags are required to have the capability of: 1) computing hash values based on given random seeds and tag ID, 2) operating segment counters for the compressed filter, 3) performing basic arithmetic operations such as subtraction and comparison, and 4) storing random seeds broadcast by the reader. Unfortunately, the current commercial EPC C1G2 tags cannot fully support these functionalities. Therefore, we are not able to implement the proposed protocol entirely by the commercial tags, which are not programmable for the operation of membership check specific to our protocol. Nonetheless, we show the EPC C1G2 protocol can be reconfigured at the reader side (without reprograming the tags) to support bitmap collection, which is a key component in Phase two of the proposed protocol. This will be the component that does not need to be reimplemented if one wants to extend todays tags for missing-tag detection. Moreover, bitmap collection is also the basis for other protocols in RFID research [1], [6], [7], [9]–[18], [27], [28].

Below we first review the existing tag ID collection protocol in the C1G2 standard [40], and then show how to reconfigure it for bitmap collection, allowing each tag to pick a slot and report its presence with one bit information, e.g., making a short transmission in the slot to make the channel busy.

A. ID Collection Protocol

According to the EPC C1G2 standard, an inventory round for tag ID collection is initialized by a reader broadcasting a `Query` command. The `Query` command is 22-bit long and includes a parameter Q, which decides the number of slots in this frame. The `Query` command is followed by an ALOHA frame, which consists of 2^Q slots during which tags can transmit responses. Upon receiving a `Query`, each tag will pick a random value s in the range $[0, 2^Q - 1]$ and load s into its slot counter. The counter is reduced by one for each `Query` or `QueryRep` and a tag will transmits back a response when the slot counter is reduced to 0.

Fig. 5 shows an ALOHA frame generated from our testbed using the ID collection protocol following the EPC C1G2 standard [40]. We use two Laird antennas [44], one as a transmitter and the other as a receiver, and we perform ID collection on the USRP (universal software radio peripheral) platform following [41], with three widely-used commercial Alien Squiggle UHF RFID tags [45]. In the experiments, we set $Q = 3$ so the frame consists of 8 slots. The physical-layer signals between the reader and the tags are shown in Fig. 5, where there is no tag response in the first slot (following the `Query` command), since initial random values of all three tags' slot counters are non-zeros. In the second slot (following the first `QueryRep` command), a short response which includes an `RN16` is received after `QueryRep`. Then an `ACK` command containing this `RN16` is broadcast. After that, the tag receiving this `RN16` transmits back a tag-ID response since the `RN16` matches its own, and the reader successfully resolves it. In the next three slots, no tag responds to the reader. When it comes to the sixth slot, the slot counters of the remaining two tags are both decreased to zero. So they both transmit a short response following the `QueryRep` command. The reader resolves an `RN16` and includes this information in the following `ACK` command. However, since this `RN16` does not match either tag's `RN16`, no tag-ID response is transmitted. As a result, the reader fails to resolve all three tags' IDs by executing one ALOHA frame due to collision. Thus, in order to collect all tag IDs, the reader has to execute additional frame(s) until all tag IDs are collected.

B. Bitmap Protocol

We modify the above C1G2 ID collection protocol for bitmap collection that is required in Phase two of our proposed missing-tag detection solution.

For our purpose, there is no need for any tag to transmit its ID, but only to report its presence in a slot. As is stated in Section II-A, the status of each slot is classified into two types: *empty slot* and *busy slot*, which will be converted to 0 and 1, respectively. The bitmap protocol can be implemented by slightly modifying the ID collection protocol as follows: An inventory round is still initialized by a reader broadcasting a `Query` command, which is followed by an ALOHA frame of 2^Q slots. During each time slot, the reader will be reconfigured to just broadcast the `QueryRep` command and not transmit the `ACK` command after it receives any short responses from tags. Each tag after receiving a `QueryRep` command will decide whether or not it shall send back a short response in the same manner as in the ID collection protocol. The reader

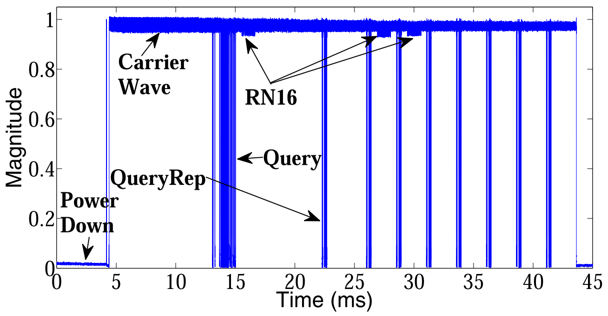


Fig. 6. Communication between a reader and tags when collecting a bitmap.

monitors the channel state and classifies a slot as a *busy* slot if it receives any short response or as an *empty* slot if it does not. After that, the reader will not broadcast an ACK command and thus no tag-ID response will be transmitted by any tag. The reader will instead initialize the next slot by transmitting QueryRep.

Fig. 6 shows an ALOHA frame generated from our testbed using the bitmap collection protocol we described above. We use the same two Laird antennas and perform the bitmap protocol on the USRP platform following [41], with three widely-used commercial Alien Squiggle UHF RFID tags [45]. We set $Q = 3$ so the frame consists of 8 slots. From Fig. 6, we can see that the first, third and fourth slot are busy slots, where short responses are received by the reader after the Query/QueryRep command, while the other five slots are empty slots. As a result, this frame is converted to a bitmap of size 8 as '10110000'.

VII. Evaluation

The proposed protocols are designed for large RFID systems with tens of thousands of tags where protocol efficiency becomes critical. For large-scale evaluation, we resort to simulations. There is limited work on missing tag detection with presence of unknown tags. Only RUN [28] and BMTD [27] 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 [28] and [27], respectively.

A. Efficiency of Compressed Filter

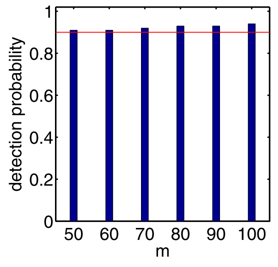
The first set of experiments study the efficiency of our compressed filter. In our simulation, we set the size of set T as 1000 and set the false positive ratios as 0.005, 0.001, 0.0005 and 0.0001 respectively. We use 10000 elements to test the actual false positive ratio of the filters.

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.

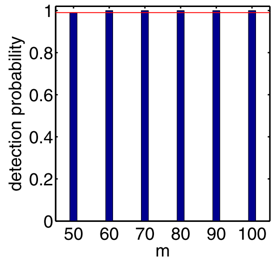
However, this does not mean that the design of original Bloom filter and optimal Bloom filter is not compact. We want to point out that for the problem of missing-tag detection with presence of unknown tags, when the required false positive ratio is very small, we can compress the Bloom filter to save 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.

TABLE I
Comparison Between the Optimal Bloom Filter and Our Compressed Filter

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



(a) α = 0.9.



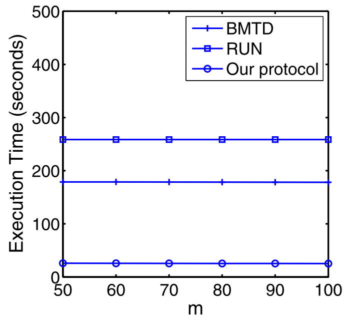
(b) α = 0.99.

Fig. 7. Reliability of our protocol.

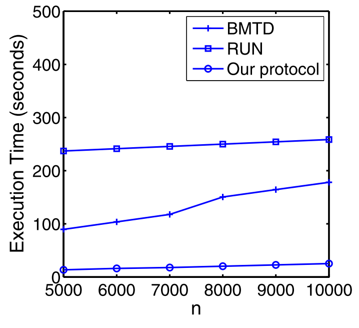
B. Accuracy

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

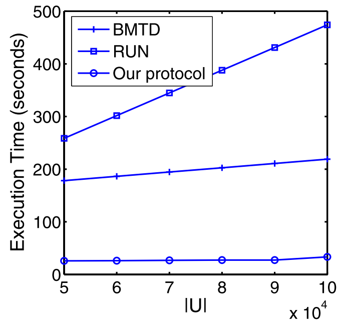
In our simulations, we set $n = 10000$, $M = 1$ and $|U| = 50000$. We vary the number m of missing tags in our system from 50 to 100 at a step size of 10 and set the detection probability α as 0.9 and 0.99 respectively. Other parameters of our protocol are optimized as is described in Section V.



(a) Impact of m.

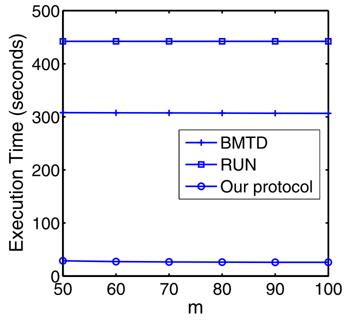


(b) Impact of n.

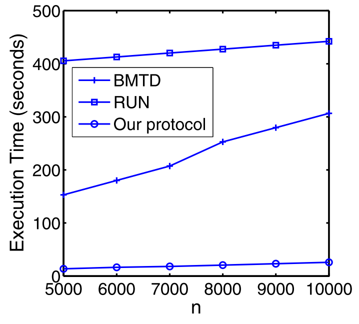


(c) Impact of $|U|$.

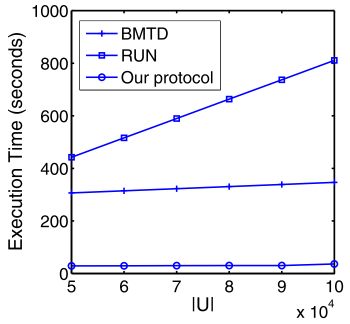
Fig. 8. Execution time of different protocols when $\alpha = 0.9$.



(a) Impact of m.



(b) Impact of n.



(c) Impact of $|U|$.

Fig. 9. Execution time of different protocols when $\alpha = 0.99$.

Fig. 7 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.

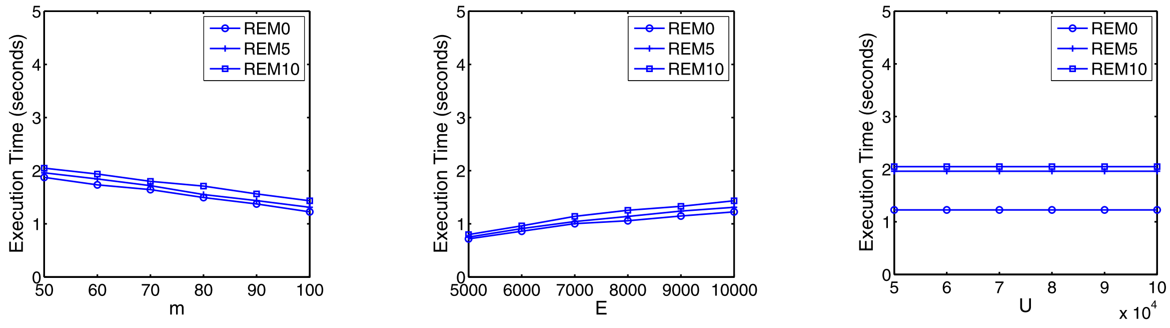
C. Execution Time

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 IV-C, we configure the EPC standard to transmit two different types of slots: T_{short} and T_{ID}, which are 2.69ms and 14.37ms respectively based on the parameter settings in [41]. As we have explained in the protocol description, slots of T_{ID} are used for transmitting the compressed filter in Phase one and the slot-index arrays in Phase two, while slots of T_{short} are used for tags to report their presence in Phase two. The parameters for our protocol are set based on Section V. 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 [28]. 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 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 [27].

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. 8a and Fig. 9a 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



(a) Impact of m. (b) Impact of n. (c) Impact of $|U|$.

Fig. 10. Execution time of our protocol in Phase two under different unreliable channels when $\alpha = 0.9$.

set $m = 100$, $|U| = 50000$ and vary n from 5000 to 10000 at a step size of 1000. The experiments are conducted under accuracy requirement of both $\alpha = 0.9$ and 0.99.

Fig. 8b and Fig. 9b 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 conducted under accuracy requirement of both $\alpha = 0.9$ and 0.99.

Fig. 8c and Fig. 9c 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.

VIII. Missing-Tag Detection Under Unreliable Channel

So far, we assume that there is no error in the communications between the RFID reader and tags and the wireless channels are reliable. However, in practice the wireless channels between a reader and a tag are not as perfect and will suffer noises and interferences from the environment, which may corrupt a time slot. For example, an empty slot may turn out to be a busy slot if it is corrupted by noises. In this case, a missing tag that is supposed to be mapped into an empty slot will be detected as present, thus reducing the detection probabilities. Besides, noises and interferences may also make an influence on busy slots. However, it is extremely unlikely in practice that the noises and the transmissions between tags and readers will have opposite phases and completely cancel each other. Therefore, as long as the reader can still detect some energy, a would-be busy slot is still detected as a busy slot. Therefore, we mainly investigate unreliable channels' impacts on empty slots in this paper.

There are many error models on unreliable channels including random error model, burst error model [46] and so on. In this paper, we mainly evaluate the impact of unreliable channels under random error model. We also stress that the random error model is only applied to the tag-to-reader communication link since passive tags harvest energy from readers and the reader-to-tag communication link is strong and much more resistent to noise than the tag-to-reader link.

A. Random Error Model

The random error model characterizes the impact of channel errors by a parameter p_e, which is the probability for each slot to be corrupted. For example, $p_e = 5\%$ indicates that a would-be empty slot has a probability of 5% to be corrupted into a busy slot by channel noises.

We now analyze the impact of random error model on our missing-tag detection protocol. Since Phase One does not involve any tag-to-reader communication, channel noise will not make an impact. For Phase Two, we know from Section V-B that in perfect channels each missing tag has a probability of $p_s \times P_{k_2}$ to be detected. Besides, under random error model the probability that a would-be empty slot is turned into a busy slot is p_e. Therefore, under random error model the detection probability in Phase Two is:

$$p_2' = 1 - (1 - p_s \times P_{k_2} \times (1 - p_e))^m. \tag{17}$$

Substituting p_2 with p_2' in Section V, we can learn the performance of our protocol under unreliable channels of random error model.

B. Evaluation Under Unreliable Channel

Since unreliable channels do not make an impact on Phase one of our protocol, we only compare the execution time of our protocol in Phase two with different error probability p_e and evaluate the impact of channel errors. We set p_e as 0, 5% and 10% and call these three protocols as REM0, REM5 and REM10, respectively. Note that REM0 is in fact our protocol under reliable channels.

In our simulation, we set $M = 1$, and let $\alpha = 0.9$. 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) as discussed in Section VII-C for evaluation. The parameters for our protocol under unreliable channels are set based on Section V.

1) Impact of m: The sixth 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 conduct under accuracy requirement of $\alpha = 0.9$.

Fig. 10a shows the results of our simulations. Similarly, the execution times of these three protocols decrease (but slowly) with the increase of m, which is expected as explained in Section VII-C.1. Besides, we can observe that our protocol takes more time under unreliable channels. As the noise level increases, the execution time needed also increases since we need a larger frame to reduce the impact of interferences from channel errors.

2) Impact of n: The seventh 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 $\alpha = 0.9$.

Fig. 10b shows the results of our simulations. The execution times of all three protocols increase with respect to n as shown in Section VII-C.2. Besides, the execution time of our protocol also increases as the noise level increases.

3) Impact of $|U|$: The eighth 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. 10c shows the results of our simulations and we can observe the similar pattern of execution time changes as explained previously.

IX. 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 a collision-reduction method to increase our efficiency. We theoretically analyze the performance of our protocol and optimize the parameters to reduce the time cost. 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.

References

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

[2] Y. Zheng, M. Li, and C. Qian, "PET: Probabilistic estimating tree for large-scale RFID estimation," in *Proc. IEEE ICDCS*, Jun. 2011, pp. 37–46.

[3] Y. Zheng and M. Li, "ZOE: Fast cardinality estimation for large-scale RFID systems," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 908–916.

[4] B. Chen, Z. Zhou, and H. Yu, "Understanding RFID counting protocols," in *Proc. Mobicom*, Sep. 2013, pp. 291–302.

[5] M. Chen, W. Luo, Z. Mo, S. Chen, and Y. Fang, "An efficient tag search protocol in large-scale RFID systems," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 899–907.

[6] S. Qi, Y. Zheng, M. Li, Y. Liu, and J. Qiu, "Scalable data access control in RFID-enabled supply chain," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 71–82.

[7] L. Xie, H. Han, Q. Li, J. Wu, and S. Lu, "Efficiently collecting histograms over RFID tags," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 145–153.

[8] M. Shahzad and A. X. Liu, "Every bit counts: Fast and scalable RFID estimation," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw. Mobicom*, 2012, pp. 365–376.

[9] L. Shangguan, Z. Zhou, X. Zheng, L. Yang, Y. Liu, and J. Han, "ShopMiner: Mining customer shopping behavior in physical clothing stores with COTS RFID devices," in *Proc. 13th ACM Conf. Embedded Netw. Sensor Syst. SenSys*, 2015, pp. 113–125.

[10] X. Liu *et al.*, "RFID cardinality estimation with blocker tags," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1679–1687.

[11] X. Liu *et al.*, "Top-k queries for categorized RFID systems," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2587–2600, Oct. 2017.

[12] Y. Hou, J. Ou, Y. Zheng, and M. Li, "PLACE: Physical layer cardinality estimation for large-scale RFID systems," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2702–2714, Oct. 2016.

[13] Q. Xiao, S. Chen, and M. Chen, "Joint property estimation for multiple RFID tag sets using snapshots of variable lengths," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Netw. Comput. MobiHoc*, 2016, pp. 151–160.

[14] M. Shahzad and A. X. Liu, "Fast and accurate tracking of population dynamics in RFID systems," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 836–846.

[15] M. Chen, S. Chen, Y. Zhou, and Y. Zhang, "Identifying state-free networked tags," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1607–1620, Jun. 2017.

[16] J. Liu, Y. Zhang, M. Chen, S. Chen, and L. Chen, "Collision-resistant communication model for stateless networked tags, poster paper," *Proc. IEEE ICNP*, Jul. 2016, pp. 221–222.

[17] Y. Zhang, S. Chen, Y. Zhou, and Y. Fang, "Anonymous temporal-spatial joint estimation at category level over multiple tag sets," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2018, pp. 846–854.

[18] Y. Zhang, S. Chen, Y. Zhou, and O. Odegbile, "Missing-tag detection with presence of unknown tags," in *Proc. 15th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2018, pp. 1–9.

[19] L. Yang, Y. Chen, X.-Y. Li, C. Xiao, M. Li, and Y. Liu, "Tagoram: Real-time tracking of mobile RFID tags to high precision using COTS devices," in *Proc. 20th Annu. Int. Conf. Mobile Comput. Netw. MobiCom*, 2014, pp. 237–248.

[20] L. Yang, Q. Lin, X. Li, T. Liu, and Y. Liu, "See through walls with COTS RFID system," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw. MobiCom*, 2015, pp. 487–499.

[21] G. Wang *et al.*, "HMRL: Relative localization of RFID tags with static devices," in *Proc. 14th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2017, pp. 1–9.

[22] C. C. Tan, B. Sheng, and Q. Li, "How to monitor for missing RFID tags," in *Proc. 28th Int. Conf. Distrib. Comput. Syst.*, Jun. 2008, pp. 295–302.

[23] T. Li, S. Chen, and Y. Ling, "Identifying the missing tags in a large RFID system," in *Proc. 11th ACM Int. Symp. Mobile ad hoc Netw. Comput. MobiHoc*, 2010, pp. 1–10.

[24] R. Zhang, Y. Liu, Y. Zhang, and J. Sun, "Fast identification of the missing tags in a large RFID system," in *Proc. 8th Annu. IEEE Commun. Soc. Conf. Sensor, Mesh Ad Hoc Commun. Netw.*, Jun. 2011.

[25] 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 Trans. Comput.*, vol. 64, no. 1, pp. 87–96, Jan. 2015.

[26] W. Luo, S. Chen, T. Li, and Y. Qiao, "Probabilistic missing-tag detection and energy-time tradeoff in large-scale RFID systems," in *Proc. 13th ACM Int. Symp. Mobile Ad Hoc Netw. Comput. MobiHoc*, 2012, pp. 95–104.

[27] J. Yu, L. Chen, R. Zhang, and K. Wang, "Finding needles in a haystack: Missing tag detection in large RFID systems," *IEEE Trans. Commun.*, vol. 65, no. 5, pp. 2036–2047, Feb. 2017.

[28] M. Shahzad and A. X. Liu, "Expecting the unexpected: Fast and reliable detection of missing RFID tags in the wild," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1939–1947.

[29] J. Ou, M. Li, and Y. Zheng, "Come and be served: Parallel decoding for COTS RFID tags," *Proc. ACM MobiCom*, 2015, pp. 500–511.

[30] J. Liu, Y. Zhang, S. Chen, M. Chen, and L. Chen, "Collision-resistant communication model for state-free networked tags," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 656–665.

[31] J. Han *et al.*, "GenePrint: Generic and accurate physical-layer identification for UHF RFID tags," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 846–858, Apr. 2016.

[32] Y. Zhang, S. Chen, Y. Zhou, Y. Fang, and C. Qian, "Monitoring bodily oscillation with RFID tags," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3840–3854, Apr. 2019.

[33] Y. Zhang, S. Chen, Y. Zhou, and Y. Fang, "Using wireless tags to monitor bodily oscillation," in *Proc. IEEE 15th Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, Oct. 2018, pp. 211–219.

[34] *National Retail Federation, National Retail Security Survey*. Accessed: Oct. 2015. [Online]. Available: https://nrf.com/resources/retail-library/national-retail-security-survey-2015

[35] A. D. Smith, A. A. Smith, and D. L. Baker, "Inventory management shrinkage and employee anti-theft approaches," *Int. J. Electron. Finance*, vol. 5, no. 3, p. 209, 2011.

[36] W. Luo, S. Chen, Y. Qiao, and T. Li, "Missing-tag detection and energy–time tradeoff in large-scale RFID systems with unreliable channels," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1079–1091, Aug. 2014.

[37] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol," in *Proc. SIGCOMM*, 1998, pp. 254–265.

[38] M. Mitzenmacher, "Compressed Bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, Oct. 2002.

[39] *CDAlgo*. Accessed: May 2004. [Online]. Available: https://en.wikipedia.org/wiki/Data_compression

[40] EPCglobal. *EPC Radio-Frequency Identity Protocols Class-1 Gen-2 UHF RFID Protocol for Communications at 860MHz-960MHz*. Accessed: Aug. 2008. [Online]. Available: http://www.epcglobalinc.org/uhfclg2

[41] M. Buettner and D. Wetherall, "A software radio-based UHF RFID reader for PHY/MAC experimentation," in *Proc. IEEE Int. Conf. RFID*, Apr. 2011, pp. 134–141.

[42] C. Qian, H. Ngan, Y. Liu, and L. M. Ni, "Cardinality estimation for large-scale RFID systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 9, pp. 1441–1454, Sep. 2011.

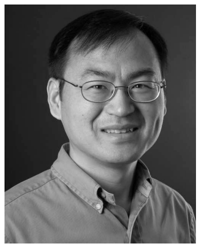
[43] M. Kodialam and T. Nandagopal, "Fast and reliable estimation schemes in RFID systems," in *Proc. 12th Annu. Int. Conf. Mobile Comput. Netw. MobiCom*, 2006, pp. 322–333.

[44] *Laird*. Accessed: May 2015. [Online]. Available: http://www.lairdtech.com/products/s9028PCL

[45] *AlienTags*. Accessed: Aug. 2005. [Online]. Available: http://www.alientechnology.com/products/tags/squiggle/

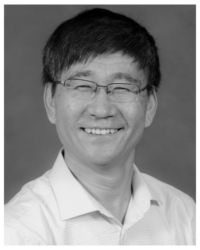
[46] B. Cornaglia and M. Spini, "Letter: New statistical model for burst error distribution," *Eur. Trans. Telecommun.*, vol. 7, no. 3, pp. 267–272, May 1996.

Youlin Zhang (Member, IEEE) received the B.S. degree in electronic information engineering from the University of Science and Technology of China, Hefei, China, in 2014. He is currently pursuing the Ph.D. degree in computer and information science and engineering with the University of Florida, Gainesville, FL, USA. His advisor is Prof. S. Chen. His research interests include big network data and the Internet of Things.



Shigang Chen (Fellow, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining University of Florida in 2002. He published more than 190 peer-reviewed journal/conference papers. He holds 12 U.S. patents. His research interests include computer networks, big data, Internet security, RFID, cyber-physical systems, and wireless communications. He received the IEEE Communications Society Best Tutorial Paper Award and the NSF CAREER Award. He is an Associate Editor of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He has served on the Editorial Board for the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, *Journal of Computer Networks*, and *ZTE Communications*. He has served in various chair positions or as a technical committee member for numerous conferences. He holds the University of Florida Research Foundation (UFRF) Professorship and the University of Florida Term Professorship from 2017 to 2020. He is an ACM Distinguished Member.

You Zhou (Member, IEEE) received the B.S. degree in electronic information engineering from the University of Science and Technology of China, Hefei, China, in 2013. He is currently pursuing the Ph.D. degree in computer and information science and engineering with the University of Florida, Gainesville, FL, USA. His advisor is Prof. S. Chen. His research interests include network security and privacy, big network data, and the Internet of Things.



Yuguang Fang (Fellow, IEEE) received the Ph.D. degree in systems engineering from Case Western Reserve University, Cleveland, OH, USA, in 1994, and the Ph.D. degree in electrical engineering from Boston University, Boston, MA, USA, in 1997.

He was an Assistant Professor with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA, from 1998 to 2000. He then joined the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA, in 2000, as an Assistant Professor, then got an early promotion to an Associate Professor with tenure in 2003 and to a Full Professor in 2005. He held the University of Florida Research Foundation (UFRF) Professorship from 2006 to 2009, the Changjiang Scholar Chair Professorship with Xidian University, Xi'an, China, from 2008 to 2011, and the Guest Chair Professorship with Tsinghua University, Beijing, China, from 2009 to 2012. He has published more than 250 articles in refereed professional journals and conferences.

Dr. Fang is also active in professional activities. He is a member of the Association for Computing Machinery (ACM). He received the National Science Foundation Faculty Early Career Award in 2001 and the Office of Naval Research Young Investigator Award in 2002, and was a recipient of the Best Paper Award in the IEEE International Conference on Network Protocols (ICNP) in 2006 and the IEEE TCGN Best Paper Award in the IEEE High-speed Networks Symposium, the IEEE GLOBECOM, in 2002. He is also serving as the Editor-in-Chief for the IEEE WIRELESS COMMUNICATIONS. He has serves/served on several editorial boards of technical journals.