How to Monitor for Missing RFID Tags

Chiu C. Tan, Bo Sheng, and Qun Li
College of William and Mary
{cct,shengbo,liqun}@cs.wm.edu

Abstract

As RFID tags become more widespread, new approaches for managing larger numbers of RFID tags will be needed. In this paper, we consider the problem of how to accurately and efficiently monitor a set of RFID tags for missing tags. Our approach accurately monitors a set of tags without collecting IDs from them. It differs from traditional research which focuses on faster ways for collecting IDs from every tag. We present two monitoring protocols, one designed for a trusted reader and another for an untrusted reader.

1 Introduction

Retail outlets lose an estimated 30 billion dollars a year to shrinkage, of which 70% are due to administration error, vendor fraud and employee theft [4]. Inexpensive RFID technology can alleviate this problem by providing a low cost and efficient means of performing inventory control. A retailer could first attach an RFID tag to each item to be monitored. Each tag contains a unique ID which is recorded and stored on a secure server. The retailer then deploys an RFID reader to periodically collect all the IDs from the tags and match them against the IDs stored on server. This way, the retailer can be immediately notified of any errors. We term this simple approach *collect all*.

However, collect all suffers from three drawbacks. First, collecting tag IDs for comparison is time consuming when there are a lot of tags due to presence of collisions. A reader collects IDs by first broadcasting the number of available time slots. Each tag will independently pick a time slot to reply. When multiple tags pick the same slot, a collision occurs and the reader obtains no information and must repeat the process again. When the set of tags is large, the number of collisions will rise, increasing the data collection time.

Second, routine monitoring usually does not require every ID to be accounted for. Consider an RFID tag attached to every product in a grocery store, and the store contains hundreds of thousands of items. In this setting, it is impractical to notify the retailer each time there is a *single* RFID tag missing. This is because a missing ID might indicate a scratched RFID tag, or simply that the RFID tag is physically blocked from receiving the query by another object. A more reasonable approach is to determine a threshold or tolerance for missing items, and alerting the retailer only when this threshold is breached.

Third, collect all is vulnerable to dishonest RFID readers returning incorrect information to the server. This is a serious threat since an estimated 45% of thefts are committed by employees [4]. A dishonest employee can first collect all the tag IDs prior to the theft, and then replay the data back to the server later.

In this paper, we consider the problem of accurately and efficiently monitoring for missing RFID tags. We assume that the RFID reader interacts with the tags and passes the collected information to the server. The server has preprogrammed threshold, and will issue a warning if the number of missing tags exceed the threshold. We provide two protocols, a trusted reader protocol (TRP) and an untrusted reader protocol (UTRP). The UTRP defends against a dishonest reader from returning inaccurate data to the server.

We make the following contributions in this paper. (1) We propose a monitoring technique which does not require the reader to collect IDs from each RFID tag, but is still able to accurately monitor for missing tags. (2) Our monitoring technique provides privacy protection by neither broadcasting tag IDs in public, nor revealing IDs to the RFID reader. (3) We present a lightweight solution to the dishonest reader problem that does not require expensive tag hardware such as an accurate on-chip timer or cryptographic MAC functions which are unavailable on passive RFID tags. (4) Our technique is more flexible than prior research in that we can accommodate different size groups of tags.

The rest of the paper is as follows. Related work is found in the next section. Section 3 contains the problem formulation, and Sections 4 and 5 present our trusted and untrusted reader protocols respectively. Section 6 evaluates our schemes, and Section 7 concludes.

2 Related Work

In an RFID system, a collision occurs when multiple tags try to transmit data to a reader at the same time. This results in the reader being unable to obtain any useful information.

Prior work [2, 3, 7, 8, 13, 15] have focused on improving protocols to reduce collisions, and secure search techniques to isolate particular tags [14] one at a time. While these techniques improve monitoring performance, such solutions are ultimately bounded by the number of tags. Regardless of the protocol used, the RFID reader will still have to isolate each tag at least once to obtain data.Our approach does not require the reader to isolate every tag.

Another approach is to use probabilistic techniques to determine some features of a large collection of RFID tags. These include methods to estimate the cardinality of a set of tags [6], and to determine popular categories of tags [12]. Our paper differs from these work by including a security protocol that deals with dishonest RFID readers.

The problem of a dishonest reader is similar to the "yoking proof" problem [5, 9, 10, 11]. A yoking proof allows an RFID reader to prove to a verifier that two RFID tags were scanned simultaneously at the same location. The yoking proof only relies on a trusted server and not a trusted RFID reader. A dishonest reader cannot tamper with the result without being detected by the sever. Bolotnyy and Robins [1] improves on the idea by creating yoking proofs for multiple tags. However,their approach requires each tag to be contacted individually *and* in a specific order. These requirements are time consuming when there are many tags. Furthermore,their scheme requires each RFID tag to have an on-chip timer that is specific to the size of the group of tags. This makes their approach inflexible in accommodating different group sizes.

3 Problem Formulation

We assume that a server has a group of objects, and an RFID tag with a unique ID is attached to each object. We refer to this group of objects as a *set of tags*. A set of tags once created is assumed to remain static, meaning no tags are added to or removed from the set.

We consider an RFID reader, R, and a set of n RFID tags, T_*. We consider this set of tags to be *"intact"* if all the tags in the set are physically present together at the same time. There are two additional parameters in our problem, a tolerance of m missing tags and a confidence level α. A set is considered intact if there are m or less tags missing. The set is considered not intact where there are at least $m + 1$ missing tags. The confidence level α specifies the lower bound of the probability that a not intact set of tags is detected. Both m and α parameters are set according to the server's requirements. A higher tolerance (m) and lower confidence level (α) will result in faster performance with less accuracy. Table 1 summerizes the remaining notations.

Anti-collision : In this paper we assume that RFID tags resolve collisions using a slotted ALOHA type scheme [15, 6]. The reader first broadcasts a frame size and a random number, (f, r), to all the tags. Each RFID tag uses the random number r and its ID to hash to a slot number sn between $[1, f]$ to return their ID, where

$$sn = h(ID \oplus r) \bmod f.$$

Tags that successfully transmit their data are instructed to keep silent. Tags that pick the same slot to reply will be informed by the reader to retransmit in subsequent rounds where the reader will send a new (f, r). The reader repeats this process until all IDs are collected.

Protocol goals : The goal of a server is to remotely, quickly, and accurately determine whether a set of tags is intact. The server first specifies a tolerance of m missing tags and a confidence level α, and instructs a reader to scan all the tags to collect a bitstring. The server then uses this result to determine whether there are any missing tags. Our protocols succeed if the server is able to determine a set of tags is not intact when more than m tags are missing with probability of at least α. In this paper, we assume that an adversary will always steal $m + 1$ tags, since for any m, the hardest scenario for the server to detect is when there are just $m + 1$ tags missing.

Adversary model : The goal of the adversary is to steal RFID tags. The adversary launches the attack by physically removing tags from the set. We do not consider more involved attacks such as "clone and replace". In such an attack, the adversary steals some tags, clones the stolen tags to make replicate tags, and replaces the replicate tags back into the set. Cloning creates replicate tags that are identical to the stolen tags. In this scenario, the server cannot detect any missing tags since the replicate tags are identical to the removed tags. This attack requires considerable technical expertise due to the cloning process, and is unlikely to be used against commodity items tracked by low cost tags.

Our paper considers two scenarios: an honest reader and a dishonest reader scenario. In the first scenario, the adversary simply attempts to steal some tags. Once the tags are stolen, the tags are assumed to be out of the range of the reader. Therefore, when a reader issues a query, the stolen tags will not reply.

In the second scenario, the adversary controls the RFID reader responsible for replying to the server. The terms "adversary" and "dishonest reader" are used interchangeably in this paper. After stealing some RFID tags, the adversary is assumed to be able to communicate with the stolen tags. This can be thought of as the adversary having a collaborator also armed with an RFID reader and the stolen tags. The adversary can communicate with the collaborator using a fast communication channel to obtain data about the stolen tags if needed.

4 TRP: Trusted Reader Protocol

In this section, we present our trusted reader protocol, TRP, where the RFID reader is assumed to be always hon-

Table 1. Notations

R/T_*	RFID reader / set of RFID tags
f/r	frame size / random number
n/m	# of tags in T_*/# of tolerated missing tags
α	confidence level
$h(.)$	hash function
sn	slot number between $[1, f]$
bs	bitstring of length f
c	number of adversary communications
ct	counter built into RFID tag

est. Given a set of RFID tags, TRP returns a bitstring to the server to check if the set of tags is intact.

4.1 Intuition and assumptions

TRP modifies the slot picking behavior used in *collect all* so that instead of having a tag pick a slot and return its ID, we let the tag simply reply with a few random bits signifying the tag has chosen that slot. In other words, instead of the reader receiving

$$\{\cdots| \quad id_1 \quad | \; 0 \; | \quad id_6 \quad | \quad collision \quad | \; 0 \; | \cdots\},$$

where 0 indicates no tag has picked that slot to reply, and $collision$ denotes multiple tags trying to reply in the same slot, the reader will receive

$$\{\cdots| \; random\ bits \; | \, 0 \, | \; random\ bits \; | \, collision \, | \, 0 \, | \cdots\}.$$

This is more efficient since the tag ID is much longer than the random bits transmitted. From the reply, the reader can generate the bitstring

$$bs = \{\cdots| \; 1 \; | \; 0 \; | \; 1 \; | \; 1 \; | \; 0 \; | \cdots\}.$$

where 1 indicates at least one tag has picked that slot.

TRP exploits the fact that a low cost RFID tag picks a reply slot in a deterministic fashion. Thus, given a particular random number r and frame size f, a tag will always pick the same slot to reply. Since the server knows all the IDs in a set, as well as the parameters (f, r), the server will be able to determine the resulting bitstring for an intact set ahead of time. The intuition behind TRP is to let the server pick a (f, r) for the reader to broadcast to the set of tags. The server then compares the bitstring returned by the reader with the bitstring generated from the server's records. A match will indicate that the set is intact.

4.2 TRP algorithm

The reader uses a different (f, r) pair each time he wants to check the intactness of T_*. The server can either communicate a new (f, r) each time the reader executes TRP, or the server can issue a list of different (f, r) pairs to the reader ahead of time.

Alg. 1 shows the overall interaction between the reader and tags. Each tag in the set executes Alg. 2 independently. The reader executes Alg. 3 to generate the bitstring bs and return it to the server. Notice that unlike the *collect all* method which requires several rounds to collect the tag information, our TRP algorithm only requires a single round. Furthermore, in Alg. 2 Line 5 the tag does not need to return the tag ID to the reader, but a much shorter random number to inform the reader of its presence. This shortens the transmission time since less bits are transmitted. bitstring to the server for verification.

Algorithm 1 Interaction between tags and R

```
1: Reader broadcasts $(f, r)$ to all tags $T_*$
2: Each tag $T_i$ executes Alg. 2
3: Reader executes Alg. 3
4: Reader returns $bs$ to server
```

Algorithm 2 Executed by Tag T_i

```
1: Receive $(f, r)$ from $R$
2: Determine slot number $sn = h(id_i \oplus r) \bmod f$
3: while $R$ broadcasts slot number do
4:     if broadcast matches $sn$ then
5:         Return random number to $R$
```

Algorithm 3 Executed by Reader R

```
1: Create bitstring $bs$ of length $f$, initialize all entries to 0
2: for slot number $sn = 1$ to $f$ do
3:     Broadcast $sn$ and listen for reply
4:     if receive reply then
5:         Set $bs[sn]$ to 1
```

4.3 Analysis

We present the analysis of how to choose a frame size f subject to a tolerance level m and confidence level α. As mentioned earlier, we define a tolerance of m missing tags, where a set of tags can be considered intact when there are at most m missing tags from the set. The set is considered not intact when at least $m + 1$ tags are missing. Since an appropriate value of m is application specific, we assume that m is a given parameter in this paper.

To quantify accuracy, we introduce a confidence parameter α. The parameter α describes the requirement of the probability of detecting at an set that is not intact. An appropriate value of α is also defined by the application. A server requiring strict monitoring can assign $m = 0$ and $\alpha = 0.99$ for high accuracy.

Our problem can be defined as given n, m and α, we want to pick the smallest f for Alg. 1 such that we can detect with more than α probability when there are more than

m out of n tags are missing. We use $g(n, x, f)$ to denote the probability of detecting the set is not intact with frame size f when exactly x tags are missing. Since the scanning time is proportional to the frame size f, our problem is formulated as to

$$\begin{aligned} &\text{minimize } f \\ s.t. \quad &\forall x > m, g(n, x, f) > \alpha. \end{aligned} \tag{1}$$

Theorem 1 *Given n, x and f,*

$$g(n, x, f) = 1 - \sum_{i=0}^{f} \binom{f}{i} p^i (1-p)^{f-i} \cdot (1 - \frac{i}{f})^x,$$

where $p = e^{-\frac{n-x}{f}}$.

PROOF. Let N_0 represent the number of empty slots in the frame generated by the currently present $n - x$ tags. A missing tag will be detected if it selects one of these N_0 slots to respond, which has a probability of $\frac{N_0}{f}$. The probability that we can not detect any of x missing tags is $(1 - \frac{N_0}{f})^x$. For each slot, the probability of being one of the N_0 empty slot is $p = (1 - \frac{1}{f})^{n-x} = e^{-\frac{n-x}{f}}$. Thus, N_0 is a random variable following a binomial distribution. For $i \in [0, f]$,

$$Pr(N_0 = i) = \binom{f}{i} p^i (1-p)^{f-i}.$$

Therefore,

$$\begin{aligned} g(n, x, f) &= 1 - \sum_{i=0}^{f} Pr(N_0 = i) \cdot (1 - \frac{i}{f})^x \\ &= 1 - \sum_{i=0}^{f} \binom{f}{i} p^i (1-p)^{f-i} \cdot (1 - \frac{i}{f})^x. \end{aligned}$$

□

Lemma 1 *Given n and f, if $x_1 > x_2$, then $g(n, x_1, f) > g(n, x_2, f)$.*

PROOF. It is obvious that more missing tags tend to yield higher probability of being detected. □

Theorem 2 *If we set $g(n, m+1, f) > \alpha$, the accuracy constraint (1) is satisfied.*

PROOF. According to Lemma 1, $\forall x > m$, $g(n, x, f) \geq g(n, m+1, f)$. Therefore, missing exactly $m+1$ tags is the worst case for our detection. Thus, any value of f satisfying $g(n, m+1, f) > \alpha$ can guarantee the accuracy requirement. □

Considering the objective, the optimal value of f is

$$f = \min\{x | g(n, m+1, x) > \alpha\}. \tag{2}$$

5 UTRP: UnTrusted Reader Protocol

In this section, we discuss UTRP, our protocol to defend against an untrusted reader. UTRP prevents a dishonest reader from generating a bs that can satisfy the server without having an intact set. For sake of brevity, the terms "dishonest reader" and "reader" are used interchangeable for the remainder of this section. An honest reader will be explicitely specified.

5.1 Vulnerabilities

In the introduction, we mentioned that a dishonest reader can replay a previously collected bitstring bs back to the server. This attack can be easily defeated by letting the server issue a new (f, r) each time the reader scans the set of tags. This renders previously collected bitstrings invalid. However, simply issuing a new (f, r) cannot defend against a dishonest reader and a collaborator.

The dishonest reader first steals a subset of tags from the original set of tags, and gives the stolen tags to his collaborator. We denote the remaining set of tags as s_1 and the stolen tags as s_2. The collaborator is also equipped with an RFID reader. The dishonest reader is denoted as R_1 and the collaborator's reader is denoted as R_2. When the server issues a new (f, r), the dishonest reader will scan the remaining set of tags s_1, and instruct his collaborator to scan the stolen tags s_2 and return the collected information. The dishonest reader will then combine the information to return a bitstring to the server. Fig. 1 illustrates the attack.

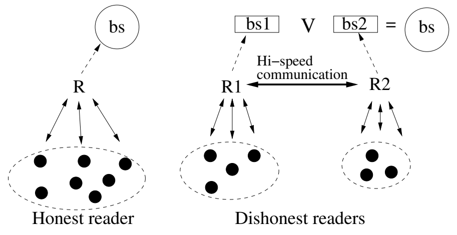

Figure 1. Vulnerability of TRP

The reader succeeds if he is able to generate a proof $\hat{bs}$ from s_1 and s_2 located in two separate locations, such that $\hat{bs}$ is the same as bs. The reader assigns himself as R_1 to read s_1 and his collaborator as R_2 to read s_2. We assume that R_1 and R_2 both know (f, r). Alg. 4 presents the algorithm of the attack. We see that so long as the both readers $R_1$and R_2 have a high speed communication, they behave just like a single reader.

One possible defense against the attack is to require a reader to complete Alg. 1 within some specified time limit t. However, selecting an appropriate value of t is difficult since t has to be long enough for an honest reader to complete a bs for the server, yet short enough such that R_1 and

Algorithm 4 Attack algorithm against TRP

```
1: Both $R_1$ and $R_2$ execute Alg. 1 on $s_1$ and $s_2$, and obtains $bs_{s_1}$ and $bs_{s_2}$ respectively.
2: $R_2$ forwards $bs_{s_2}$ to $R_1$.
3: $R_1$ executes $(bs_{s_1} \vee bs_{s_2})$ to obtain $\hat{bs}$, where $\hat{bs} = bs$
4: $R_1$ returns $\hat{bs}$ to the server.
```

R_2 cannot collaborate by passing data to each other. For instance, in Alg. 4, R_1 and R_2 can derive a correct $\hat{bs}$ by just having *one* transmission. Assuming that R_1 and R_2 communicates via a high speed channel, estimating a time limit t that is shorter than the time needed for a single transmission is difficult.

5.2 Intuition and assumptions

The intuition behind our solution is to force collaborating readers to communicate multiple times such that the latency is large enough to be accurately estimated by the server. UTRP accomplishes this by introducing two additional components, a re-seeding process, and a counter.

Each time a reader encounters a time slot that is chosen by at least one RFID tag, UTRP requires the reader to *re-seed* by sending a new (f, r) to all tags that have yet to reply. The new f is equal to the number of slots left from the previous f. For example, initially we have $f = 10$ and the first slot has a tag reply. The new f value will thus be 9. The new random number r is determined by the server. The re-seeding will result in a bs different from the prior one. We illustrate an example in Fig. 2. We let the tag $T1$ to be the first tag to reply. The reader will send a new (f, r) to remaining tags to pick a new slot. Tag $T2$ picks a different slot after re-seeding, creating a different bs. Collaborating readers wanting to obtain $\hat{bs} = bs$ have to re-seed each time either reader receives a reply. Since no reader can determine in advance which slot will receive a reply, collaborating readers must check with each other after either reader obtains a reply in a single slot.

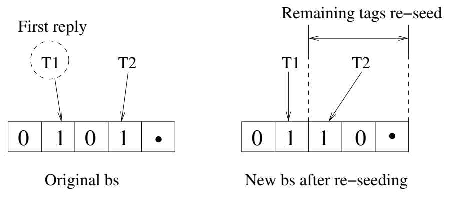

Figure 2. Re-seeding after first reply

However, re-seeding does not prevent readers from running the algorithm multiple times to gain some information. Each reader can first read every slot in frame size f to determine which slot has a reply. The readers then exchange this information and scan all the tags again to arrive at the correct bitstring. For example in Fig. 3, R_1 and R_2 first scan all their tags to determine that a re-seed is necessary in slot 2. Both readers can then repeat the process by re-seeding tags starting from slot 2 to complete the bs. A mechanism to prevent a reader from going backwards is needed.

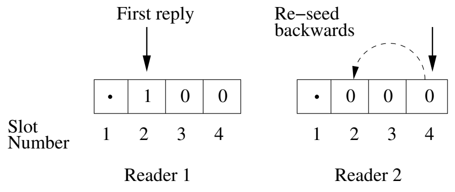

Figure 3. Re-seeding just from slot 2

We adopt an assumption made in several earlier research [1, 5, 9, 10, 11] that each RFID tag has a counter ct, and the counter will automatically increment each time the tag receives a new (f, r) pair. A reader that attempts to move backwards to re-seed the tags will have an incorrect counter value. An RFID tag now picks a slot as

$$sn = h(ID \oplus r \oplus ct) \bmod f.$$

Recall that the server knows the ID, and provides the frame size f and random numbers r. The server also knows the value of each tag's counter ct since ct only increments when queried by the reader. Thus, the server can still determine the correct bs for verification.

Algorithm 5 Interaction between server and R

```
1: Server generates $(f, r_1, \cdots, r_f)$, sends to $R$, and starts the timer
2: $R$ broadcasts $(f, r_1)$ to all the tags $T_*$
3: $T_*$ executes Alg. 7
4: $R$ executes Alg. 6
5: if $R$ returns correct $bs$ to server before timer expires then
6:     Server verifies $R$'s proof
```

Algorithm 6 UTRP algorithm for reader R

```
1: Create a bitstring $bs$ of length $f$, initialize all entries to 0.
2: Set $f' = f$
3: for slot number $sn = 1$ to $f$ do
4:     Broadcast $sn - f + f'$ and listen for reply
5:     if receive reply then
6:         Set $bs[sn]$ to 1, and $f' = f - sn$
7:         Broadcast $(f', r)$ where $r$ is the next random number in the sequence
8: Return $bs$ to server
```

5.3 UTRP algorithms

We let the server issue a frame size together with f random numbers, $(f, r_1, \cdots, r_f)$, to a reader. The reader is

Algorithm 7 UTRP algorithm for tag T_i

```
1: Receive $(f, r)$ from $R$. Increment $ct = ct + 1$.
2: Determine slot number $sn = h(id_i \oplus r \oplus ct)$ mod $f$
3: while $R$ is broadcasting do
4:     if $R$ broadcasts slot number and slot number matches $sn$ then
5:         Return random number to $R$, keep silent
6:     else if $R$ broadcasts a new frame size and random number $(f, r)$ then
7:         Receive $(f, r)$ from $R$. Increment $ct = ct + 1$
8:         Determine new slot number $sn = h(id_i \oplus r \oplus ct)$ mod $f$
```

supposed to use each random number only once in the given order. For example, let $f = 15$ and $r_1 = 5, r_2 = 9$. Reader R will first send out $(15, 5)$. Assuming that some tag replies in the first slot, R is supposed to re-seed by broadcasting $(14, 9)$ so that each remaining tag can pick a new slot. A reader that does not follow this rule will not yield the right answer to the server.

Alg. 5 illustrates the overall protocol, and Alg. 6 and Alg. 7 show the reader and tag behavior respectively. Collaborating readers will have to communicate with each other after Alg. 6 Line 5 to determine whether to re-seed. If either collaborating reader receives a reply, both readers must re-seed. A reader cannot predict in advance whether any tag will reply in the next slot since a tag picks a slot number sn using the random number r, and the list of random numbers is determined by the server.

The reader also cannot attempt to execute Alg. 6 multiple times to determine which slots will have a reply since the counter value will change. In Alg. 7 Line 1, the tag will automatically increment the counter each time it receives a new (f, r). Since a tag in UTRP picks a new slot using $ID \oplus r \oplus ct$, a different ct will cause the final bs to be different. Since an RFID tag can only communicate with a single reader at a time, the counter in Alg. 7 will not be incremented by any other readers.

5.4 Analysis

The analysis for UTRP is similar to the TRP analysis presented earlier. The difference is that in TRP, the information contained in the missing tags is gone. In UTRP, we consider the dishonest R removes more than m missing tags, but yet is able to obtain some information from the removed tags. Compared with TRP, when the same number of tags are missing, the dishonest reader has higher probability to pass the verification since the dishonest reader has more information than that in TRP.

UTRP requires the reader to return bs before timer t expires. The intuition here is to limit the communication between dishonest readers, thus increase the probability of detecting the missing tags. The communication time increases with the number of readers an adversary controls, making it easier for an adversary to be detected. In our analysis, we consider the best case for an adversary to escape detection by having the adversary only control *two* readers.

For a given frame size and random number, the scanning time for a honest reader to finish the protocol may vary. The server sets the timer to an empirical value, which is conservative so that a honest reader can definitely respond before the due time. We assume that the server can estimate the minimum and maximum scanning time of a honest reader, indicated as ST_{min} and ST_{max} respectively. The server thus sets $t = ST_{max}$.

Since a reader cannot predict in advance in which slot there will be a reply, UTRP forces the dishonest readers to wait for a message from other readers every time it encounters an empty slot. If a dishonest reader receives a reply in the current slot, it can continue re-seeding and scanning the following slots without waiting for the results from other readers. We let t_{comm} be the average communication overhead between two dishonest readers. Given t, we claim that the dishonest readers can communicate in at most $c = \frac{t - ST_{min}}{t_{comm}}$ slots.

Let us consider the whole set of n tags is divided into two sets s_1 and s_2. Without loss of generality, let $|s_1| \geq |s_2| > m$. Assume there are two dishonest readers R_1 and R_2 scanning s_1 and s_2 respectively. Each time R_1 encounters an empty slot (a slot where no tag replies), R_1 will have to pause to check with R_2. If R_2 receives a reply in *that* particular slot, both R_1 and R_2 will have to re-seed. Otherwise R_1 can continue broadcasting the remaining slots. Since the dishonest readers cannot communicate after every slot within t, the best strategy for the dishonest readers to pass our verification is as follows: (1) R_1 waits for the messages from R_2 in the first c empty slots it has encountered; (2) R_1 finishes scanning the following slots (with s_1) and sends the bitstring to the server.

With this strategy, the first part (with communication) of the bitstring is correct, but the remaining part may be suspicious. The following analysis tries to derive an appropriate value for f, such that the server can catch the difference in this scenario with high probability ($> \alpha$).

Similar to the TRP analysis, the worst case occurs when the number of missing tags is just beyond the tolerant range, i.e., $|s_2| = m + 1$. Intuitively, while the number of missing tags is smaller, we need longer frame size to guarantee the same accuracy requirement. In the following, we will discuss how to set parameter in this worst case to satisfy the accuracy requirement. The optimal frame size for the worst case is thus the optimal for all cases.

Theorem 3 *Assume after c' slots, the dishonest read R_1 will have encountered c number of empty slots. The ex-*

pected value of c' is $\frac{c}{e^{-\frac{n-m-1}{f}}}$.

PROOF. For each slot, the probability that no tags respond is $p = (1-\frac{1}{f})^{|s_1|} = e^{-\frac{|s_1|}{f}}$. After c' slots, the expected number of empty slots is $p \cdot c'$. By resolving $p \cdot c' = c$, we have $c' = \frac{c}{e^{-\frac{n-m-1}{f}}}$. $\square$

Theorem 4 *Let x be the number of the tags in s_2, which respond after the first c' slots. Given $i \in [0, m+1)$,*

$$Pr(x=i) = \left(\begin{array}{c} m+1 \\ i \end{array} \right) (1-\frac{c'}{f})^i (\frac{c'}{f})^{m+1-i}.$$

PROOF. Since each tag randomly picks a slot in the frame, it has $1-\frac{c'}{f}$ probability to respond after the first c slots. Thus, x follows a binomial distribution as $x \sim B(1-\frac{c'}{f}, |s_2|)$. Thus, we have

$$Pr(x=i) = \left(\begin{array}{c} m+1 \\ i \end{array} \right) (1-\frac{c'}{f})^i (\frac{c'}{f})^{m+1-i}.$$

$\square$

With similar proof, we have the following theorem.

Theorem 5 *Let y be the number of the tags in s_1, which respond after the first c' slots. Given $i \in [0, n-m-1)$,*

$$Pr(y=i) = \left(\begin{array}{c} n-m-1 \\ i \end{array} \right) (1-\frac{c'}{f})^i (\frac{c'}{f})^{n-m-i-1}.$$

On one hand, in s_2, the tags replying after the first c' slots are 'real' missing tags in this problem. On the other hand, among the tags in s_1, only those responding after the first c' slots are considered useful in detecting the missing tags. For a given frame size f, $f - c'$ is the effective frame size for distinguishing the bitstring with missing tags. Thus, the server has $g(x+y, x, f-c')$ probability to detect the difference. Considering all possible values of x and y, a frame size f can satisfy the accuracy requirement, if

$$\sum_{i=0}^{m+1} \sum_{j=0}^{n-m-1} Pr(x=i) \cdot Pr(y=j) \cdot g(i+j, i, f-c') > \alpha. \quad (3)$$

Therefore, the optimal frame size is the minimal value satisfying the above condition.

6 Evaluation

In this paper, we use simulations to evaluate the efficiency and accuracy of TRP and UTRP. We measure efficiency by the frame size f. A smaller f has fewer slots, which translates into faster performance. We assume the duration of each slot is equally long. We measure accuracy by first setting values of m and α to derive a f satisfying Eq. (2) for TRP and Eq. (3) for UTRP. We then execute our simulation to test if our protocols can determine "missing" when there are $m+1$ tags randomly removed from the set. We average the results over 1000 trials.

We perform simulations varying n from 100 to 2000 tags at 100 tag increments. The tolerance level is set to tolerate m = 5, 10, 20 and 30 missing tags. Finally, we uniformly set our confidence $\alpha = 0.95$.

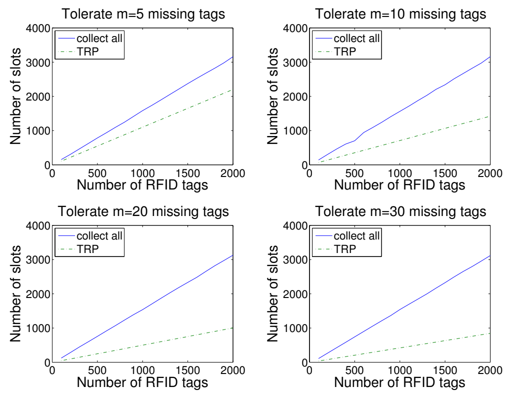

Figure 4. Comparing *collect all* versus TRP

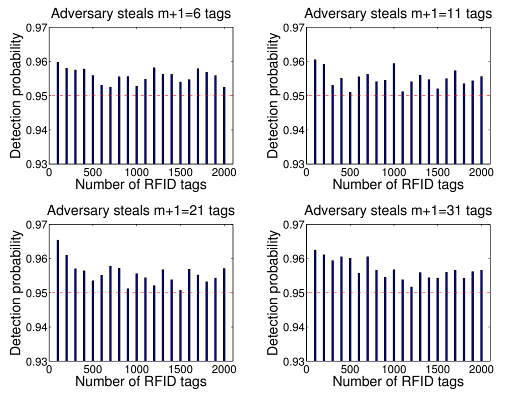

Figure 5. Accuracy of TRP with $\alpha = 0.95$

Fig. 4 compares the efficiency between the *collect all* method against our TRP algorithm. Lee et. al. [7] determined that the optimal frame size is equal to the number of unidentified tags in a set. Based on this, we simulate *collect all* by setting $f = n$ in the first round, and f equal to the remaining tags that have yet to transmit. The final number of slots for *collect all* method is the sum of all the fs used in each round. To accommodate the tolerance m, we consider *collect all* algorithm to be completed once $n - m$ tags are collected. From Fig. 4, we observe that the scanning time in both *collect all* and TRP increases linearly as the number of tags increases. TRP uses fewer slots, especially when the set size is large. Note that the actual performance of *collect all* will be worse since the tag needs to return its ID rather than a shorter random number in TRP, resulting in a longer duration of each slot.

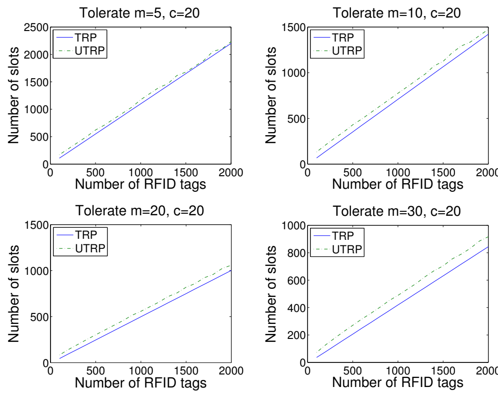

Figure 6. Comparing TRP versus UTRP

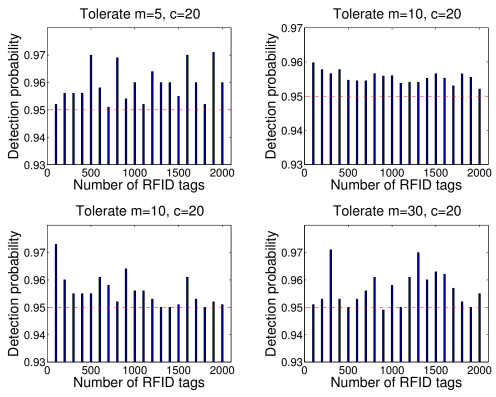

Figure 7. Accuracy of UTRP with $\alpha = 0.95$

Fig. 5 shows the accuracy of TRP when using the frame size f shown in Fig. 4. With a tolerance of m, the most difficult situation for TRP to detect missing tags is when there are just $m + 1$ missing tags. The horizontal dashed line in Fig. 5 denotes the confidence level α. Each bar represents probability TRP detects $m+1$ missing tags from a set. Bars over the horizontal line denotes TRP has successful detected $m + 1$ missing tags with probability greater than α. As we can see TRP detects the missing tags over probability α. In evaluating UTRP, we assume that a dishonest reader splits the set into two, and can communicate for $c = 20$ slots before executing Alg. 5 on the remaining tags in his set. To determine the efficiency of UTRP, we compare the size of f used in UTRP against TRP, and Fig. 6 shows the results. We observe that the overhead of UTRP over TRP is small. Note that for UTRP, we have added a very small number of slots (between 5 10 slots) to the the optimal frame size given in Eq. (3). This is because the derivation of c' in Theorem 3 relies on the expected value, which introduces a slight inaccuracy. Note that Fig. 6 does not imply that the performance of UTRP is comparable to TRP since we do not take into account the time needed for R to broadcast a new (f, r) pair to remaining RFID tags in UTRP. Finally, the accuracy of UTRP is shown in Fig. 7. UTRP also accurately detects missing tags with probability larger than the confidence level α.

7 Conclusion

In this paper, we consider the problem of monitoring for missing RFID tags. We provide protocols for both an honest and dishonest RFID reader. Our approach differs from prior work in that our techniques do not require the reader to collect the ID from every tag.

Acknowledgments

We would like to thank the reviewers for their helpful comments. This project was supported by US National Science Foundation award CCF-0514985, CNS-0721443, and CNS-0747108.

References

[1] L. Bolotnyy and G. Robins. Generalized “Yoking-Proofs” for a group of RFID tags. In *Mobiquitous 2006*.

[2] M. A. Bonuccelli, F. Lonetti, and F. Martelli. Tree slotted ALOHA: a new protocol for tag identification in RFID networks. In *WoWMoM 2006*.

[3] J.-R. Cha and J.-H. Kim. Novel anti-collision algorithms for fast object identification in RFID system. In *ICPADS 2005*.

[4] R. Hollinger and J. Davis. National retail security survey 2001.

[5] A. Juels. “Yoking-Proofs” for RFID tags. In *Pervasive Computing and Communications Workshops 2004*.

[6] M. Kodialam and T. Nandagopal. Fast and reliable estimation schemes in RFID systems. In *MobiCom 2006*.

[7] S.-R. Lee, S.-D. Joo, and C.-W. Lee. An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification. In *Mobiquitous 2005*.

[8] A. Micic, A. Nayak, D. Simplot-Ryl, and I. Stojmenovic. A hybrid randomized protocol for RFID tag identification. In *WoNGeN 2005*.

[9] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda. Solving the simultaneous scanning problem anonymously: Clumping proofs for RFID tags. In *SecPerU 2007*.

[10] S. Piramuthu. On existence proofs for multiple RFID tags. In *ICPS 2006*.

[11] J. Saito and K. Sakurai. Grouping proof for RFID tags. In *AINA 2005*.

[12] B. Sheng, C. C. Tan, Q. Li, and W. Mao. Finding popular categories for rfid tags. In *Mobihoc 2008*.

[13] D. Simplot-Ryl, I. Stojmenovic, A. Micic, and A. Nayak. A hybrid randomized protocol for RFID tag identification. In *Sensor Review 2006*.

[14] C. C. Tan, B. Sheng, and Q. Li. Severless search and authentication protocols for rfid. In *International Conference on Pervasive Computing and Communications (PerCom)*, 2007.

[15] H. Vogt. Efficient object identification with passive RFID tags. In *Pervasive 2002*.