Efficient Techniques for Monitoring Missing RFID Tags

Chiu C. Tan, Bo Sheng, and Qun Li

***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 the other for an untrusted reader.**

***Index Terms*—Authentication, privacy, RFID, search, security.**

I. Introduction

INVENTORY control is one of the main applications of radio frequency identity (RFID) technology. By attaching an RFID tag on items to be monitored, the owner can periodically scan the tags to determine what tags, and hence what items, are currently present. This process is useful for instance, to retailers wanting to detect shrinkage (thief), a serious problem estimated to cost 30 billion dollars a year [1].

This simple method of scanning all the tags and compare them against earlier records poses two problems. First, as the number of RFID tags becomes large, collecting every tag id is very time consuming. While existing research efforts have focused on improving hardware and protocol design to speed up data collection, the increasing number of RFID tags will inevitably outpace our ability to collect the data. Walmart is estimated to generate seven terabytes of RFID data per day when RFID tags are attached to individual items [2]. Simply collecting all the data is no longer feasible. We argue that a different approach for monitoring large numbers of tags is needed.

Second, the straightforward approach cannot defend against a dishonest RFID reader. A dishonest reader can simply collect the tag ids prior to the theft, and replay the data back to the server later. The conventional defense against a dishonest reader centers on creating "yoking proofs" [3] where a set of RFID tags create a chain of dependent MAC computations [4] within a given time. However, yoking proofs require additional tag improvements like an accurate on-chip timer and cryptographic MAC functions which increase the cost of the tag. The on-chip timer is also very inflexible since different number of tags will need different amounts of time to complete a proof, but the on-chip timer once designed onto hardware cannot be modified. A flexible solution suitable for low cost RFID tags is desired.

In this paper, we consider the problem of accurately and efficiently monitoring a large group of RFID tags for missing tags. We consider the scenario that the RFID reader can interact with the tags and pass the collected information to the server so that the server will be able to remotely monitor the "intactness" of the group of tags. We provide two protocols to solve this problem, a trusted reader protocol and a untrusted reader protocol. In the first protocol, a short bitstring is formed by forcing each tag to reply in a certain slot in the reply frame. The bitstring can be a very efficient way to determine whether a certain number of tags are missing. In the second protocol, in order to solve the "yoking" problem in a more flexible fashion, we adopt a timer in the server side and force two readers reading two split sets to communicate with each other frequently so that the timer may expire for an invalid reading.

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) We present a lightweight solution that resistant to a dishonest RFID reader returning inaccurate answers. (3) We improve on [5] by performing evaluations using parameters derived from commercial RFID hardware. These results as well as additional discussion on practical details such as channel conditions can better discern the actual impacts of these protocols.

The rest of the paper is organized as follows. We review the related work in the next section, followed by our problem formulation. Sections IV and V contains trusted and untrusted reader protocols respectively. We evaluate our schemes in Section VI, and discuss some practical implementation issues in Section VII. Finally, we conclude in Section $VIII$.

II. Related Work

The idea behind improving the performance of data collection from RFID tags usually involves reducing the number of collisions between tags. In an RFID system, an reader uses a slotted ALOHA-like scheme to regulate tag replies. Under this scheme [6], [7], 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 together with its own id to hash to a slot number sn between $[1, f]$ to return their id, where $sn = h(id \oplus r) \bmod f$. The reader broadcast the end of each slot to the tags and each tag will decrement their sn accordingly. When a tag's sn reaches slot 0, the tag will

Manuscript received October 24, 2008; revised June 29, 2009 and November 29, 2009; accepted April 5, 2010. The associate editor coordinating the review of this paper and approving it for publication was G. Mandyam.

C. C. Tan and Q. Li are with the Department of Computer Science, College of William and Mary, Williamsburg, VA, 23187 (e-mail: {cct, liqun}@cs.wm.edu).

B. Sheng is with the College of Computer and Information Science, Northeastern University, Boston, MA, 02115 (e-mail: shengbo@ccs.neu.edu).

This project was supported by the US National Science Foundation award CNS-0721443, CNS-0831904, CAREER Award CNS-0747108.

Digital Object Identifier 10.1109/TWC.2010.06.081301

broadcast a 16 bit random number (RN16). For instance, assuming we have two RFID tags, A and B, which pick slots numbers 2 and 3 respectively.

Reader broadcast	Tag A slot number	Tag B slot number	Note
0^{th} broadcast	2	3	
1^{st} broadcast	1	2	
2^{nd} broadcast	0	1	Tag A responds
3^{rd} broadcast	NA	0	Tag B responds

If no other tags' sn is also slot 0, the reader will receive only one RN16, and respond with an ACK, after which the tag will reply with its id. If multiple tags broadcast RN16s, the reader will detect a collision and not send an ACK. The tags will not respond with their ids, but keep silent waiting for the next (f, r) pair. The reader can repeat this process, which we term *collect all*, to identify all the tags in the set. Many schemes [8]–[13] have been proposed to reduce collisions so as to improve the data collection process.

While faster data collection will definitely improve monitoring performance, these solutions are ultimately bounded by the number of tags. Regardless of the protocol used, the RFID reader will still have to isolate each tag in the collection at least once to obtain data. We take a different approach which does not require the reader to isolate every tag.

The problem of using a dishonest reader is similar to the "yoking proof" problem introduced by Juels [3]. A yoking proof allows an RFID reader to prove to a verifier that two RFID tags were scanned simultaneously at the same location. The proof cannot be tampered by an adversarial reader. Later work by [14], [15] and [16] improves on this idea, but is still limited to *two* tags.

More relevant to our dishonest reader problem is [4] which creates a proof for an arbitrary number of RFID tags. The paper assumes that all RFID tags are queried by the reader in a pre-specified order. The reader first sends a value to the first RFID tag to sign with its secret key. Let us call the return value a_1. The reader then forwards a_1 to the second tag to sign, and collects the result a_2. The reader sends a_2 to the third tag, and so on until all tags are contacted. The reader can use the final result to prove that all the tags are present. Each RFID tag is assumed to have an on-chip timer and incrementing counter. The counter is factored into each tag's encryption, and is engineered to automatically increment when the timer expires. When the reader first starts the proof, the timer is initialized. If the reader does not complete the proof in time, the counter will increment. Subsequent RFID tag's encryption will have a different counter value, resulting in an invalid proof.

However, [4] requires the reader to contact each RFID tag individually, a time consuming process when there are a large number of tags. In addition, [4] requires the reader to query the set of RFID tags in a specific order to generate a correct proof. This requirement creates additional overhead since the order in which tags are read is irrelevant when monitoring for theft. Finally, their paper assumes that each RFID tag has an on-chip timer, which is inflexible in accommodating different group sizes.

TABLE I
NOTATIONS

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

Another possible defense against dishonest readers is to use cryptographic search protocols like [17] where the RFID tags will maintain a timestamp of the previous successful read. A secure server can issue an access list to the RFID reader that is valid only for a given time period, and require the reader to return the set of collected RFID tag responses. A dishonest reader that tries to query the tags multiple times ahead of time will be detected by the server. Cryptographic search protocols share a similar limitation as yoking proof protocols in that the reader must contact each tag one at a time, leading to bad performance when the number of tags to be monitored is large.

III. 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 added to or removed from the set.

We consider an RFID reader, R, entrusted with a set of n RFID tags, T_*, by the server. In this paper, 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. Both parameters are set according to the server's requirements. A higher tolerance and lower confidence level will result in faster performance with less accuracy. A set is considered intact if there are m or less tags missing. The confidence level α specifies the lower bound of the probability that the missing tags are detected. Table I summerizes the notation used in this paper.

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 where an adversary steals some tags, clones the stolen tags to make replicate tags, and replaces the replicate tags back into the set. Under this scenario, the server cannot detect any missing tags if the replicate tags are identical to the removed

tags. Since this complicated attack requires certain technical expertise, and is unlikely to be used against commodity items tracked by low cost tags, we do not consider this form of attacks in this paper.

Our paper considers two scenarios, the first where the reader contacted by the server is honest, and the second where the reader is dishonest. 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" can be used interchangeably in this scenario. After stealing some RFID tags, the adversary is assumed to still 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. The stolen tags are in the control of the collaborator. The adversary can communicate with the collaborator using a fast communication channel to obtain data about the stolen tags if needed.

IV. TRP: Trusted Reader Protocol

In this section, we present our trusted reader protocol,TRP, where the RFID reader is assumed to be always honest. Given a set of RFID tags, TRP returns a bitstring to the server to check if the set of tags is intact.

A. 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 picked that slot to reply, and $collision$ denotes multiple tags trying to reply in the same slot, the reader will receive

$$\{\cdots|\text{random bits}| \ 0 \ |\text{random bits}|\text{random bits}| \ 0 \ |\cdots\}.$$

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.

B. 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. The tag can return a much shorter random number to inform the reader of its presence. This shortens the transmission time since less bits are transmitted.

Algorithm 1 Interaction between reader R and group of tags T_*

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 Algorithm for 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 Algorithm for Reader R

1: Create bitstring array 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

C. Analysis

In this subsection 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 least $m+1$ missing tags. 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,

$$g(n,x,f) = 1 - \sum_{i=0}^{f} \left(\begin{array}{c} f \\ i \end{array} \right) p^i (1-p)^{f-i} \cdot (1 - \frac{i}{f})^x,$$

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 $i \in [0, f]$, $Pr(N_0 = i) = \left(\begin{array}{c} f \\ i \end{array} \right) p^i (1-p)^{f-i}$. Therefore,

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

■

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 $g(n, m+1, f) > \alpha$, then 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 of minimizing f, the optimal value of f is

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

V. 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.

A. Vulnerabilities

We begin by examining how TRP is vulnerable when executed by an untrusted reader. As mentioned earlier, a dishonest reader can reply previously collected bitstring bs back to the server. The server can attempt to prevent such an attack by issuing a new (f, r) to the reader every time the reader executes TRP. However, using a new (f, r) will not foil a dishonest reader with access to multiple colluding readers linked by a high speed communication channel. In this attack, the reader first divides the set of tags into smaller groups, and assigns each smaller group to his collaborator. Each collaborator is assumed to have an RFID reader. When the reader wants to convince the server the set is intact, the reader's collaborators can scan their respective groups and forward the data to the reader. For simplicity, we assume that the reader divides the original set into two groups, s_1 and s_2. Fig. 1 illustrates the attack.

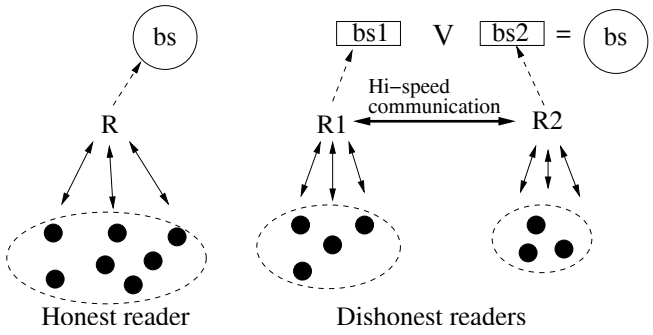

Fig. 1. Vulnerability of TRP.

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, R_1 and R_2 behave just like a single reader.

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.

A simple solution is to require a reader to complete Alg. 1 within some specified time t. However, selecting an appropriate 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 R_2 cannot collaborate by passing data to each other. For instance, in Alg. 4, a correct $\hat{bs}$ can be derived if R_2 can communicate with R_1 less than t. Since a successful attack requires only *one* communication between the readers, picking a good t is difficult.

B. Intuition and assumptions

The intuition behind our solution is to force collaborating readers to communicate enough 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.

UTRP requires a reader to *re-seed* by sending a new (f, r) to all tags that have yet to reply each time the reader

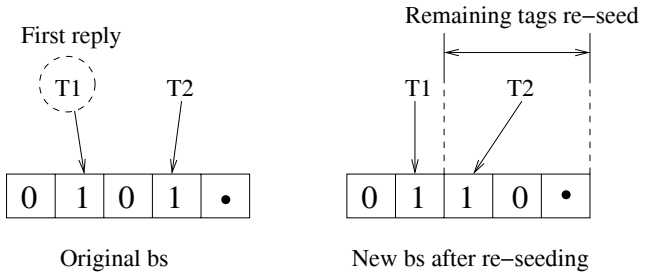

Fig. 2. Re-seeding after first reply.

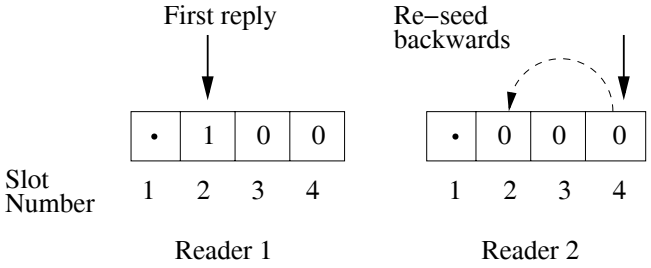

Fig. 3. Re-seeding just from slot 2.

encounters a 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. To illustrate, in Fig. 2, tag $T1$ is the first 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.

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.

We adopt an assumption made in several earlier research [3], [4], [14]–[16] that each RFID tag has a counter ct, and the value of the counter can only be incremented by one each time. The tag will automatically increment its counter each time it receives a (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 $sn = h(id \oplus r \oplus ct) \bmod f$.

C. 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 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.

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 accepts R's proof

Algorithm 6 UTRP algorithm for reader R

1: Create a bitstring array 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

Algorithm 7 UTRP algorithm for tag T_i

1: Receive (f, r) from R. Increment $ct = ct + 1$.
2: Determine slot number $sn = h(id_i \oplus r \oplus ct) \bmod f$
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 $sn = h(id_i \oplus r \oplus ct) \bmod f$

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. Note that 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.

D. 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 of using a timer is to limit the communication between dishonest readers, thus increase the probability of detecting the missing tags. 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 expected 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 $c' = \frac{c}{e^{-\frac{n-m-1}{f}}}$. ■

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

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

■

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.

VI. Evaluation

We use extensive simulations to evaluate the efficiency and accuracy of TRP and UTRP. To determine the performance, we need to estimate the latency of different types of slots. We use the reported values for the Alien ALR-9800 EPC Class 1 Gen 2 RFID reader depicted in [18] to estimate the latency of a collision slot, as well as the amount of time needed to transmit the RFID tag's id back to the reader. We assume that the time needed for an empty slot as equal to the time needed for a collision slot. While a reader can determine an slot is empty without waiting for the entire time duration needed to transmit an RN16, different types of RFID readers may wait for different amounts of time. This assumption serves as an upper bound, since a reader can determine whether the slot is empty, single, or collision after waiting for a period of time equal to that needed to transmit an RN16.

A. TRP Performance

We compare TRP against the conventional *collect all* method. We implemented the *collect all* method using the guidelines proposed by [9] that set the optimal frame size to the number of unidentified tags left in a set, that is $f = n$ in the first round, and f set to equal the remaining tags that have yet to transmit for subsequent rounds. Note this implementation

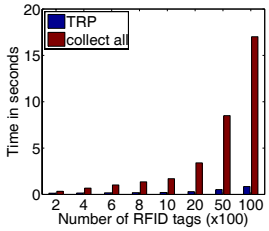

Fig. 4. TRP vs *collect all*.

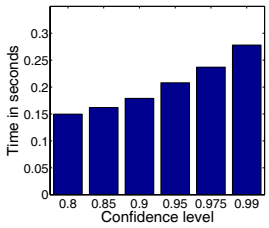

Fig. 5. Time to complete TRP with different α values.

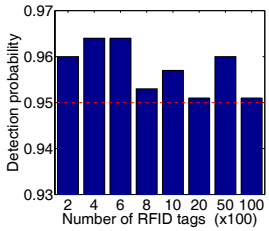

Fig. 6. Accuracy of TRP ($\alpha = 0.95$, averaged over 1000 trials).

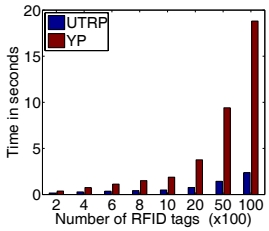

Fig. 7. UTRP vs YP.

of *collect all* considers an ideal scenario where a reader is assumed to know the cardinality of remaining tags to compute the optimal frame size. An RFID reader cannot do this in practice. To accommodate the tolerance m, we consider *collect all* algorithm to be completed once $n - m$ tags are collected.

We perform simulations varying the number of tags n from 200 to 10000 tags, and setting the tolerance level m to one percent of the n number of tags. Fig. 4 compares the efficiency between our TRP algorithm and the *collect all* method with the confidence level $\alpha = 0.95$. We observe that when the number of tags are small, approximately 200 tags, the performance of *collect all* and TRP is about the same. However, when the number of tags to monitor increase, the time latency of TRP grows much slower than the time needed for *collect all*. When monitoring 10000 RFID tags, TRP is approximately 20 times faster than *collect all*, requiring only 0.8 seconds to complete verses 17 seconds for *collect all*. We show the results of changing the confidence level α in Fig. 5 at 1000 tags. The additional time needed to when selecting higher values of α, for example, from 0.9 to 0.95 is very small, only 0.4 seconds. These results show that TRP is suitable for monitoring large number of tags.

Fig. 6 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. 6 denotes the confidence level α. Each bar represents probability TRP detects $m + 1$ missing tags from a set. Bars over the horizontal line denote where 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.

B. UTRP Performance

The closest alternative to defending against a dishonest reader is to use yoking proofs. We use the yoking proof for arbitrary number of tags [4], which we denote as YP, to compare against UTRP. Since YP cannot be executing by existing RFID equipment, we estimate it's performance as follows. YP requires the reader to query the RFID tags one at a time in a fixed order. Each RFID tag will execute a cryptographic function and return the answer back to the reader for further processing. Using [18], we assume that the time needed query just one tag is 1.9 ms. This time is approximately the time needed for the reader to broadcast the tag id and the tag to respond with its id. This is only an approximation because the RFID reader mode necessary read just one tag without doing anti-collision was not performed in [18]. We estimate the cryptographic overhead for a reader and each tag to be 0.1 ms. Finally, we assume that the last m tags are chosen as the m tolerated missing tags.

To evaluate UTRP, we let a dishonest reader split the set of tags into two, and assume that the dishonest reader can communicate with his accomplice for $c = 20$ slots before executing Alg. 5 on the remaining tags in his set. To determine the efficiency of UTRP, we compare the time needed to execute UTRP against YP in Fig. 7. We see that UTRP outperforms YP for all group sizes of RFID tags. In terms of accuracy, UTRP like TRP is able to detect the missing tags with probability larger than the confidence level α. The difference is that in UTRP, we add a very small number of slots (between 5 and 10 slots) to the calculated 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.

VII. Practical Discussion

The evaluation in the prior section demonstrated the performance gains from TRP and UTRP. In this section, we examine the practical issues involved with implementing TRP or UTRP on actual hardware. We begin by examining the tradeoffs of our protocols against existing solutions.

Protocol tradeoffs: The alternative of TRP is the *collect all* method. The tradeoff for faster performance is that TRP does not return as much information as *collect all*. TRP is unable, for instance, determine how many tags are missing, not what are the missing tag ids. As such, a user can use TRP to determine when to do an expensive *collect all* operation. Given that TRP is almost 20 times faster when dealing with large tag populations, a user can save considerable time by running TRP frequently to detect thief, and only execute *collect all* periodically to collect more detailed information.

A similar tradeoff exists when comparing UTRP against YP, in that UTRP is faster but returns less information. However, we argue that unlike the TRP and *collect all*, UTRP is a better choice than YP for dealing with an untrusted reader. YP requires a fixed timer to be built into each RFID tag, and this timer is configured to timeout depending the size of the group of tags. This means that the a tag group of different sizes will need a different timer value. The cost of changing this timer value when tags are reorganized is too high, making YP inflexible to accommodate different group sizes.

Implementation issues: While both TRP and UTRP do not require a significant departure from existing RFID standards, some modifications are still needed for implementation. First, both protocols require the RFID tags only broadcast the

random RN16 number, and not their actual ids. This can be accomplished by programming the RFID reader to always assume that each reply slot in the bitstring is a collision. This will "fool" the tags into thinking there is a collision and not return their ids. Second, both protocols require the reader to output to the user the results of the bitstring after the tags have replied. Current commercial readers like the Alien ALR-9800 do collect this information, but do not provide the means of returning it to the user. A more open reader platform [19] may allow this function to be implemented in the future. Finally, UTRP requires the RFID tags to contain an incremental counter which current Class 1 Gen 2 RFID tags do not implement. A slightly more advance RFID tag can be implemented to provide this function.

Environmental conditions: In a practical setting, environmental conditions such as background noise and absorption rates of different materials, will affect the wireless channel between the reader and tags. To illustrate, we set up an experiment by placing an RFID tag on an empty glass bottle, and place the reader 50 cm away. The reader is programmed to continuously read the tag for several seconds. This allows us to determine the *read rate*, the number of times the tag was read per second. This is one of the standard metrics for RFID read performance [20]. In this setup, we were able to obtain a read rate of 14.67 reads per second. We then filled up the bottle with water, and repeated the experiment again, and obtaining a read rate of 13.8 reads per second. This shows the effects of the medium on RFID tag performance.

We can mitigate this effect using two strategies. The first is to change the tolerance level m to accommodate the missing tag replies. The user can experimentally estimate the number of responses that will be missed and adjust m accordingly. For example, let the experiments indicate about 1% of the present RFID tags will not respond, and the user is willing to tolerate 2% missing RFID tags. Then the user will set $m = 2\% + 1\% * (100\% - 2\%) = 2.98\%$.

The second strategy is to place multiple RFID tags onto the same object. The intuition is that at certain angles, an RFID tag may not be read by the RFID reader. Placing multiple tags at different locations on the same object makes it more likely that at least one of the tags will be picked up by the RFID reader [21]. We let each RFID tag ID take the form {Group ID|Tag ID}. The RFID reader will query each group ID separately, perform a logical OR on the returned bitstring from each group, and run the protocols on the resulting bitstring. By placing multiple tags on the same object, even if one of the tags is blocked and cannot received the reader's signal, the other tags can still respond when the reader tests for a different group. This way, our schemes will not erroneously conclude there are missing tags when in effect there are not.

VIII. Conclusion

In this paper, we consider the problem of monitoring a large set of RFID tags for missing tags. We provide two protocols,TRP for an honest reader, and UTRP for a dishonest reader, to accurately and efficiently monitor the RFID tags. Our results show that TRP and UTRP performs 20 and 8 times faster than the alternative methods respectively when there are 10000 tags in a given set. At the same time, both protocols are able to accurately detect tags are missing even when there is only one more missing tag than the tolerance level. These results suggests that our protocols are especially suitable for large numbers of RFID tags.

Acknowledgment

The authors would like to thank the reviewers for all their helpful comments. This project was supported by US NSF award CNS-0721443, CNS-0831904, CAREER Award CNS-0747108.

References

[1] R. Hollinger and J. Davis, "National retail security survey," 2001.
[2] H. Gonzalez, J. Han, X. Li, and D. Klabjan, "Warehousing and analyzing massive RFID data sets," *ICDE 2006*.
[3] A. Juels, ""Yoking-Proofs" for RFID tags," in *Pervasive Comput. Commun. Workshops*, 2004.
[4] L. Bolotnyy and G. Robins, "Generalized "Yoking-Proofs" for a group of RFID tags," in *International Conf. Mobile Ubiquitous Syst.*, 2006.
[5] C. C. Tan, B. Sheng, and Q. Li, "How to monitor for missing RFID tags," *IEEE ICDCS*, 2008.
[6] M. Kodialam and T. Nandagopal, "Fast and reliable estimation schemes in RFID systems," *ACM Mobicom*, 2006.
[7] M. Kodialam, T. Nandagopal, and W. C. Lau, "Anonymous tracking using RFID tags," *IEEE Infocom 2007*.
[8] D. Simplot-Ryl, I. Stojmenovic, A. Micic, and A. Nayak, "A hybrid randomized protocol for RFID tag identification," *Sensor Rev.*, 2006.
[9] S.-R. Lee, S.-D. Joo, and C.-W. Lee, "An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification," in *International Conf. Mobile Ubiquitous Syst.*, 2005.
[10] M. A. Bonuccelli, F. Lonetti, and F. Martelli, "Tree slotted ALOHA: a new protocol for tag identification in RFID networks," *IEEE WoWMoM*, 2006.
[11] H. Vogt, "Efficient object identification with passive RFID tags," *Pervasive*, 2002.
[12] J.-R. Cha and J.-H. Kim, "Novel anti-collision algorithms for fast object identification in RFID system," *IPDPS*, 2005.
[13] A. Micic, A. Nayak, D. Simplot-Ryl, and I. Stojmenovic, "A hybrid randomized protocol for RFID tag identification," *WoNGeN*, 2005.
[14] 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," *SecPerU*, 2007.
[15] S. Piramuthu, "On existence proofs for multiple RFID tags," in *IEEE International Conf. Pervasive Services*, 2006.
[16] J. Saito and K. Sakurai, "Grouping proof for RFID tags," *Advanced Inf. Netw. Applications (AINA)*, 2005.
[17] T. Y. Won, J. Y. Chun, and D. H. Lee, "Strong authentication protocol for secure rfid tag search without help of central database," *Embedded Ubiquitous Computing, IEEE/IFIP International Conf.*, 2008.
[18] M. Buettner and D. Wetherall, "An emperical study of UHF RFID performance," *ACM Mobicom*, 2008.
[19] [Online]. Available: http://www.epcglobalinc.org/standards/llrp/, "EPC low level reader protocol (LLRP) standard."
[20] S. Aroor and D. Deavours, "Evaluation of the state of passive UHF RFID: an experimental approach," *IEEE Syst. J.*, vol. 1, 2007.
[21] L. Bolotnyy and G. Robins, "Multi tag RFID systems," *Int. J. Internet Protoc. Technol.*, 2007.

Chiu C. Tan is currently a graduate student at Department of Computer Science, College of William and Mary. His research interests include ubiquitous computing, embedded systems, and large scale RFID systems.

Bo Sheng is a postdoc at Northeastern University. He received his Ph.D. in computer science from the College of William and Mary in 2010. His research interests include wireless networks and embedded systems.

Qun Li Qun Li is an associate professor in the Department of Computer Science at the College of William and Mary. He holds a PhD degree in computer science from Dartmouth College.