Chapter 2
Efficient Tag Search in Large RFID Systems

This chapter introduces the tag search problem in large RFID systems. A new technique called filtering vector is designed to reduce the transmission overhead during search process, thereby improving the time efficiency. Based on this technique, we present an iterative tag search protocol. Some tags are filtered out in each round and the search process will eventually terminate when the result meets a given accuracy requirement. Moreover, the protocol is extended to work under noisy channel. The simulation results demonstrate that our protocol performs much better than the best existing work.

The rest of this chapter is organized as follows. Section 2.1 gives the system model and the problem statement. Section 2.2 briefly introduces the related work. Section 2.3 describes our new protocol in detail. Section 2.4 addresses noisy wireless channel. Section 2.5 evaluates the performance of our protocol by simulations. Section 2.6 gives the summary.

2.1 System Model and Problem Statement

2.1.1 System Model

We consider an RFID system consisting of one or more readers, a backend server, and a large number of tags. Each tag has a unique 96-bit ID according to the EPC global Class-1 Gen-2 (C1G2) standard [9]. A tag is able to communicate with the reader wirelessly and perform some computations such as hashing. The backend server is responsible for data storage, information processing, and coordination. It is capable of carrying out high-performance computations. Each reader is connected to the backend server via a high speed wired or wireless link. If there are many readers (or antennas), we divide them into non-interfering groups and any RFID protocol can be performed for one group at a time, with the readers in that group executing the protocol in parallel. The readers in each group can be regarded as an integrated unit, still called a reader for simplicity. Many works regarding multi-reader coordination can be found in literature [5, 7, 17].

In practice, the tag-to-reader transmission rate and the reader-to-tag transmission rate may be different and subject to the environment. For example, as specified in the EPC global Class-1 Gen-2 standard, the tag-to-reader transmission rate is 40–640kbps in the FM0 encoding format or 5–320kbps in the Miller modulated subcarrier encoding format, while the reader-to-tag transmission rate is about 26.7–128kbps. However, to simplify our discussions, we assume the tag-to-reader transmission rate and the reader-to-tag transmission rate are the same, and it is straightforward to adapt our protocol for asymmetric transmission rates.

2.1.2 Time Slots

The RFID reader and the tags in its coverage area use a framed slotted MAC protocol to communicate. We assume that clocks of the reader and all tags in the RFID system are synchronized by the reader's signal. During each frame, the communication is initialized by the reader in a request-and-response mode, namely the reader broadcasts a request with some parameters to the tags and then waits for the tags to reply in the subsequent time slots.

Consider an arbitrary time slot. We call it an empty slot if no tag replies in this slot, or a busy slot if one or more tags respond in this slot. Generally, a tag just needs to send one-bit information to make the channel busy such that the reader can sense its existence. The reader uses "0" to represent an empty slot with an idle channel and "1" for a busy slot with a busy channel. The length of a slot for a tag to transmit a one-bit short response is denoted as t_s. Note that t_s can be set larger than the time of one-bit data transmission for better tolerance of clock drift in tags. Some prior RFID work needs another type of slots for transmission of tag IDs, which will be introduced shortly.

2.1.3 Problem Statement

Suppose we are interested in a known set of tag IDs $X = \{x_1, x_2, x_3, \cdots\}$, each $x_i \in X$ is called a *wanted tag*. For example, the set may contain tag IDs on a certain type of products under recall by a manufacturer. Let $Y = \{y_1, y_2, y_3, \cdots\}$ be the set of tags within the coverage area of an RFID system (e.g., in a warehouse). Each x_i or y_i represents a tag ID. The tag search problem is to identify the subset W of wanted tags that are present in the coverage area. Namely, $W \subseteq X$. Since each tag in W is in the coverage area, $W \subseteq Y$. Therefore, $W = X \cap Y$. We define the intersection ratio of X and Y as

$$R_{INTS} = \frac{|W|}{\min\{|X|, |Y|\}}. \tag{2.1}$$

Exactly finding W can be expensive if X and Y are very large. It is much more efficient to find W approximately, allowing small bounded error [28]—all wanted tags in the coverage area must be identified, but a few wanted ones that are not in the coverage may be accidentally included.[1]

Our solution performs iteratively. Each round rules out some tags in X when it becomes certain that they are not in the coverage area (i.e., Y), and it also rules out some tags in Y when it becomes certain that they are not wanted ones in X. These ruled-out tags are called non-candidate tags. Other tags that remain possible to be in both X and Y are called candidate tags. At the beginning, the search result is initialized to all wanted tags X. As our solution is iteratively executed, the search result shrinks towards W when more and more non-candidates are ruled out.

Let W^* be the final search result. We have the following two requirements:

1. All wanted tags in the coverage area must be detected, namely $W \subseteq W^*$.
2. A false positive occurs when a tag in $X - W$ is included in W^*, i.e., a tag not in the coverage area is kept in the search result by the reader.[2] The false-positive ratio is the probability for any tag in $X - W$ to be in W^* after the execution of a search protocol. We want to bound the false-positive ratio by a pre-specified system requirement P_{REQ}, whose value is set by the user. In other words, we expect

$$\frac{|W^* - W|}{|X - W|} \leq P_{REQ}. \tag{2.2}$$

Notations used in this chapter are given in Table 2.1 for quick reference.

2.2 Related Work

2.2.1 *Tag Identification*

A straightforward solution for the tag search problem is identifying all existing tags in Y. After that, we can apply an intersection operation $X \cap Y$ to compute W. EPC C1G2 standard assumes that the reader can only read one tag ID at a time. Dynamic Framed Slotted ALOHA (DFSA) [4, 8, 19–21] is implemented to deal with tag collisions, where each frame consists of a certain number of equal-duration slots.

[1] If perfect accuracy is necessary, a post step may be taken by the reader to broadcast the identified IDs. As the wanted tags in the coverage reply after hearing their IDs, those mistakenly included tags can be excluded due to non-response to these IDs.

[2] The nature of our protocol guarantees that all tags in $Y - W$ are not included in W^*.

Table 2.1 Notations

Symbols	Descriptions
X	Set of wanted tags
Y	Set of tags in the RFID system
W	Intersection of X and Y, i.e., $W = X \cap Y$
X_i	Set of remaining candidate tags in X, i.e., search result at the beginning of the ith round of our protocol;
Y_i	Set of remaining candidate tags in Y at the beginning of the ith round of our protocol
U_i	Difference between X_i and W, i.e., $U_i = X_i - W$
V_i	Difference between Y_i and W, i.e., $V_i = Y_i - W$
$\lvert \cdot \rvert$	Cardinality of the set
$h(\cdot)$	A uniform hash function
$FV(\cdot)$	Filtering vector of a set

It is proved that the theoretical upper bound of identification throughput using DFSA is approximately $\frac{1}{e}$ tags per slot (e is the natural constant), which is achieved when the frame size is set equal to the number of unidentified tags [25]. As specified in EPC C1G2, each slot consists of the transmissions of a QueryAdjust or QueryRep command from the reader, one tag ID, and two 16-bit random numbers: one for the channel reservation (collision avoidance) sent by the tags, and the other for ACK/NAK transmitted by the reader. We denote the duration of each slot for tag identification as t_l. Therefore, the lower bound of identification time for tags in Y using DFSA is

$$T_{DFSA} = e \times |Y| \times t_l. \tag{2.3}$$

One limitation of the current DFSA is that the information contained in collision slots is wasted. Some recent work [3, 12, 15, 16, 24, 27] focuses on Collision Recovery (CR) techniques, which enable the resolution of multiple tag IDs from a collision slot. Benefiting from the CR techniques, the identification throughput can be dramatically improved up to 3.1 tags per slot in [16]. Suppose the throughput is υ tags per slot after adopting the CR techniques. The lower bound for identification time is

$$T_{CR} = \frac{|Y|}{\upsilon} \times t_l. \tag{2.4}$$

Note that after employing the CR techniques the real duration of each slot can be longer than t_l. The reason is that the reader may need to acknowledge multiple tags and the tags may need to send extra messages to facilitate collision recovery.

2.2.2 Polling Protocol

The polling protocol provides an alternative solution to the tag search problem. Instead of collecting all IDs in Y, the reader can broadcast the IDs in X one by one. Upon receiving an ID, each tag checks whether the received ID is identical to its own. If so, the tag transmits a one-bit short response to notify the reader about its presence; otherwise, the tag keeps silent. Hence, the execution time of the polling protocol is

$$T_{Polling} = |X| \times (t_{id} + t_s), \tag{2.5}$$

where t_{id} is the time cost for the reader to broadcast a tag ID.

The polling protocol is very efficient when $|X|$ is small. However, it also has serious limitations. First, it does not work well when $|X| \gg |Y|$. Second, the energy consumption of tags (particularly when active tags are used) is significant because tags in Y have to continuously listen to the channel and receive a large number of IDs until its own ID is received.

2.2.3 CATS Protocol

To address the problems of the tag identification and polling protocols, Zheng et al. design a two-phase protocol named *Compact Approximator based Tag Searching protocol* (CATS) [28], which is the most efficient solution for the tag search problem to date.

The main idea of the CATS protocol is to encode tag IDs into a Bloom filter and then transmit the Bloom filter instead of the IDs. In its first phase, the reader encodes all IDs of wanted tags in X into an L_1-bit Bloom filter, and then broadcasts this filter together with some parameters to tags in the coverage area. Having received this Bloom filter, each tag tests whether it belongs to the set X. If the answer is negative, the tag is a non-candidate and will keep silent for the remaining time. After the filtration of phase one, the number of candidate tags in Y is reduced. During the second phase, the remaining candidate tags in Y report their presence in a second L_2-bit Bloom filter constructed from a frame of time slots t_s. Each candidate tag transmits in k slots that it is mapped to. Listening to channel, the reader builds the Bloom filter based on the status of the time slots: "0" for an idle slot where no tag transmits, and "1" for a busy slot where at least one tag transmits. Using this Bloom filter, the reader conducts filtration for the IDs in X to see which of them belong to Y, and the result is regarded as $X \cap Y$.

With a pre-specified false-positive ratio requirement P_{REQ}, the CATS protocol uses the following optimal settings for L_1 and L_2:

$$L_1 = |X| \log_{\phi}\left(-\frac{\alpha|X|}{\beta|Y| \ln P_{REQ}}\right), \tag{2.6}$$

$$L_2 = \frac{|X|}{\ln \phi}\left(\ln P_{REQ} - \frac{\alpha}{\beta}\right), \tag{2.7}$$

where ϕ is a constant that equals 0.6185, α and β are constants pertaining to the reader-to-tag transmission rate and the tag-to-reader transmission rate, respectively. In CATS, the authors assume t_s is the time needed to delivering one-bit data, and $\alpha = \beta$, i.e., the reader-to-tag transmission rate and the tag-to-reader transmission rate are identical. Therefore, the total search time of the CATS protocol is

$$\begin{aligned} T_{CATS} &= (L_1 + L_2) \times t_s \\ &= |X|\left(\log_{\phi}\left(\frac{-|X|}{|Y| \ln P_{REQ}}\right) + \frac{\ln P_{REQ} - 1}{\ln \phi}\right) \times t_s. \end{aligned} \tag{2.8}$$

2.3 A Fast Tag Search Protocol Based on Filtering Vectors

This section presents an Iterative Tag Search Protocol (ITSP) to solve the tag search problem in large-scale RFID systems. We will ignore channel error for now and delay this subject to Sect. 2.4.

2.3.1 Motivation

Although the CATS protocol takes a significant step forward in solving the tag search problem, it still has several important drawbacks. First, when optimizing the Bloom filter sizes L_1 and L_2, CATS approximates $|X \cap Y|$ simply as $|X|$. This rough approximation may cause considerable overhead when $|X \cap Y|$ deviates significantly from $|X|$.

Second, it assumes that $|X| < |Y|$ in its design and formula derivation. In reality, the number of wanted tags may be far greater than the number in the coverage area of an RFID system. For example, there may be a huge number $|X|$ of tagged products that are under recall, but as the products are distributed to many warehouses, the number $|Y|$ of tags in a particular warehouse may be much smaller than $|X|$. Although CATS can still work under conditions of $|X| >> |Y|$, it will become less efficient as our simulations will demonstrate.

Third, the performance of CATS is sensitive to the false-positive ratio requirement P_{REQ}. The performance deteriorates when the value of P_{REQ} is very small. While the simulations in [28] set $P_{REQ} = 5\ \%$, its value may have to be much smaller in some practical cases. For example, suppose $|X| = 100{,}000$, and $|W| = 1000$. If we set $P_{REQ} = 5\,\%$, the number of wanted tags that are falsely claimed to be in Y by CATS will be up to $|X - W| \times P_{REQ} = 4995$, far more than the 1000 wanted tags that are actually in Y.

We will show that an iterative way of implementing Bloom filters is much more efficient than the classical way that the CATS protocol adopts.

2.3.2 *Bloom Filter*

A Bloom filter is a compact data structure that encodes the membership for a set of items. To represent a set $S = \{e_1, e_2, \cdots, e_n\}$ using a Bloom filter, we need a bit array of length l in which all bits are initialized to zeros. To encode each element $e \in S$, we use k hash functions, $h_1, h_2, \cdots, h_k$, to map the element randomly to k bits in the bit array, and set those bits to ones. For membership lookup of an element b, we again map the element to k bits in the array and see if all of them are ones. If so, we claim that b belongs to S; otherwise, it must be true that $b \notin S$. A Bloom filter may cause false positive: a non-member element is falsely claimed as a member in S. The probability for a false positive to occur in a membership lookup is given as follows [2, 23]:

$$P_B = \left(1 - \left(1 - \frac{1}{l}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/l}\right)^k. \tag{2.9}$$

When $k = \ln 2 \times \frac{l}{n}$, P_B is approximately minimized to $\left(\frac{1}{2}\right)^k = \left(\frac{1}{2}\right)^{\ln 2 \frac{l}{n}}$. In order to achieve a target value of P_B, the minimum size of the filter is $-\frac{\ln P_B}{(\ln 2)^2} n$.

CATS sends one Bloom filter from the reader to tags and another Bloom filter from tags back to the reader. Consider the first Bloom filter that encodes X. As $n = |X|$, the filter size is $-\frac{\ln P_B}{(\ln 2)^2} |X|$. As an example, to achieve $P_B = 0.001$, the size becomes $14.4 \times |X|$ bits. Similarly, the size of the second filter from tags to the reader is also related to the target false-positive probability.

Below we show that the overall size of the Bloom filter can be significantly reduced by reconstructing it as filtering vectors and then iteratively applying these vectors.

2.3.3 *Filtering Vectors*

A Bloom filter can also be implemented in a segmented way. We divide its bit array into k equal segments, and the ith hash function will map each element to a random bit in the ith segment, for $i \in [1...k]$. We name each segment as a filtering vector (FV), which has l/k bits. The following formula gives the false-positive probability of a single filtering vector, i.e., the probability for a non-member to be hashed to a "1" bit in the vector:

$$P_{FV} = 1 - \left(1 - \frac{1}{l/k}\right)^n \approx 1 - e^{-kn/l}. \tag{2.10}$$

Since there are k independent segments, the overall false-positive probability of a segmented Bloom filter is

$$P_{FP} = (P_{FV})^k \approx \left(1 - e^{-kn/l}\right)^k, \tag{2.11}$$

which is approximately the same as the result in (2.9). It means that the two ways of implementing a Bloom filter have similar performance. The value P_{FP} is also minimized when $k = \ln 2 \times \frac{l}{n}$. Hence, the optimal size of each filtering vector is

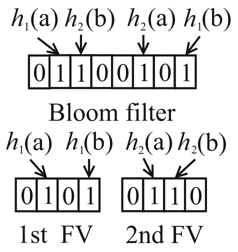
$$\frac{l}{k} = \frac{n}{\ln 2}, \tag{2.12}$$

which results in

$$P_{FV} \approx \frac{1}{2}. \tag{2.13}$$

Namely, each filtering vector on average filters out half of non-members.

Figure 2.1 illustrates the concept of filtering vectors. Suppose we have two elements a and b, two hash function h_1 and h_2, and an 8-bit bit array. First, suppose $h_1(a)$ mod 8 = 1, $h_1(b)$ mod 8 = 7, $h_2(a)$ mod 8 = 5, $h_2(b)$ mod 8 = 2, and we construct a Bloom filter for a and b in the upper half of the figure. Next, we divide the bit array into two 4-bit filtering vectors, and apply h_1 to the first segment and h_2 to the second segment. Since $h_1(a)$ mod 4 = 1, $h_1(b)$ mod 4 = 3, $h_2(a)$ mod 4 = 1, $h_2(b)$ mod 4 = 2, we build the two filtering vectors in the lower half of the figure.



h_1(a) h_2(b) h_2(a) h_1(b)
0 1 1 0 0 1 0 1
Bloom filter
h_1(a) h_1(b) h_2(a) h_2(b)
0 1 0 1
0 1 1 0
1st FV
2nd FV

Fig. 2.1 Bloom filter and filtering vectors

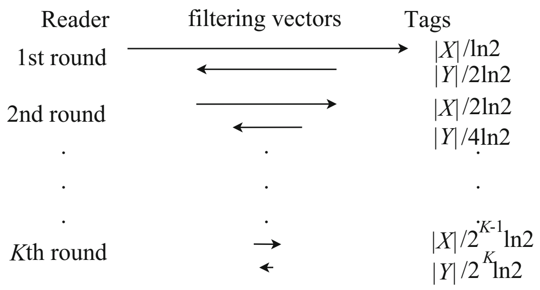


Fig. 2.2 Iterative use of filtering vectors. Each *arrow* represents one filtering vector, and the *length of the arrow* indicates the filtering vector's size, which is specified to the right. As the size shrinks in subsequent rounds, the total amount of data exchanged between the reader and the tags is significantly reduced

2.3.4 Iterative Use of Filtering Vectors

In this work, we use filtering vectors in a novel iterative way: Bloom filters between the reader and tags are exchanged in rounds; one filtering vector is exchanged in each round, and the size of filtering vector is continuously reduced in subsequent rounds, such that the overall size of each Bloom filter is much reduced.

Below we use a simplified example to explain the idea, which is illustrated in Fig. 2.2: Suppose there is no wanted tag in the coverage area of an RFID reader, namely $X \cap Y = \emptyset$. In round one, we firstly encode X in a filtering vector of size $|X|/\ln 2$ through a hash function h_1, and broadcast the vector to filter tags in Y. Using the same hash function, each candidate tag in Y knows which bit in the vector it is mapped to, and it only needs to check the value of that bit. If the bit is zero, the tag becomes a non-candidate and will not participate in the protocol execution further. The filtering vector reduces the number of candidate tags in Y to about $|Y| \times P_{FV} \approx |Y|/2$. Then a filtering vector of size $|Y|/(2\ln 2)$ is sent from the remaining candidate tags in Y back to the reader in a way similar to [28]: Each candidate tag hashes its ID to a slot in a time frame and transmit one-bit response in that slot. By listening to the states of the slots in the time frame, the reader constructs the filtering vector, "1" for busy slots and "0" for empty slots. The reader uses this vector to filter non-candidate tags from X. After filtering, the number of candidate tags remaining in X is reduced to about $|X| \times P_{FV} \approx |X|/2$. Only the candidate tags in X need to be encoded in the next filtering vector, using a different hash function h_2. Hence, in the second round, the size of the filtering vector from the reader to tags is reduced by half to $|X|/(2\ln 2)$, and similarly the size of the filtering vector from tags to the reader is also reduced by half to $|Y|/(4\ln 2)$. Repeating the above process, it is easy to see that in the i_{th} round, the size of the filtering vector from the reader to tags is $|X|/(2^{i-1}\ln 2)$, and the size of the filtering vector from tags to the reader is $|Y|/(2^i \ln 2)$. After K rounds, the total size of all filtering vectors from the reader to tags is

$$\frac{1}{\ln 2}\sum_{i=1}^{K}\frac{|X|}{2^{i-1}} < \frac{2|X|}{\ln 2}, \tag{2.14}$$

where $\frac{2|X|}{\ln 2}$ is an upper bound, regardless of the number K of rounds (i.e., regardless of the requirement on the false-positive probability). It compares favorably to CATS whose filter size, $-\frac{\ln P_B}{(\ln 2)^2}|X|$, grows inversely in P_B, and reaches $14.4 \times |X|$ bits when $P_B = 0.001$ in our earlier example.

Similarly, the total size of all filtering vectors from tags to the reader is

$$\frac{1}{\ln 2}\sum_{i=1}^{K}\frac{|Y|}{2^{i}} < \frac{|Y|}{\ln 2}, \tag{2.15}$$

and $P_{FP} = (P_{FV})^K \approx \left(\frac{1}{2}\right)^K$. We can make P_{FP} as small as we like by increasing n, while the total transmission overhead never exceeds $\frac{1}{\ln 2}(2|X| + |Y|)$ bits. The strength of filtering vectors in bidirectional filtration lies in their ability to reduce the candidate sets during each round, thereby diminishing the sizes of filtering vectors in subsequent rounds and thus saving time. Its power of reducing subsequent filtering vectors is related to $|X - W|$ and $|Y - W|$. The more the numbers of tags outside of W, the more they will be filtered in each round, and the greater the effect of reduction.

2.3.5 Generalized Approach

Unlike the CATS protocol, our iterative approach divides the bidirectional filtration in tag search process into multiple rounds. Before the ith round, the set of candidate tags in X is denoted as X_i ($\subseteq X$), which is also called the search result after the $(i-1)$th round. The final search result is the set of remaining candidate tags in X after all rounds are completed. Before the ith round, the set of candidate tags in Y is denoted as Y_i ($\subseteq Y$). Initially, $X_1 = X$ and $Y_1 = Y$. We define $U_i = X_i - W$ and $V_i = Y_i - W$, which are the tags to be filtered out. Because W is always a subset of both X_i and Y_i, we have

$$\begin{aligned} |U_i| &= |X_i| - |W| \\ |V_i| &= |Y_i| - |W|. \end{aligned} \tag{2.16}$$

Instead of exchanging a single filtering vector at a time, we generalize our iterative approach by allowing multiple filtering vectors to be sent consecutively. Each round consists of two phases. In phase one of the ith round, the RFID reader broadcasts a number m_i of filtering vectors, which shrink the set of remaining candidate tags in Y from Y_i to Y_{i+1}. In phase two of the ith round, one filtering

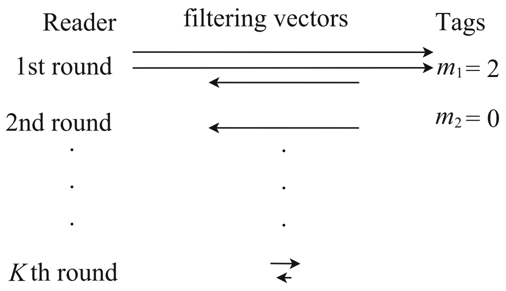


Fig. 2.3 Generalized approach. Each round has two phases. In phase one, the reader transmits zero, one, or multiple filtering vectors. In phase two, the tags send exactly one filtering vector to the reader. In the example shown by the figure, $m_1 = 2$ and $m_2 = 0$, which means there are two filtering vectors sent by the reader in the first round, while no filtering vector from the reader during the second round

vector is sent from the remaining candidate tags in Y_{i+1} back to the reader, which uses the received filtering vector to shrink its set of remaining candidates from X_i to X_{i+1}, setting the stage for the next round. This process continues until the false-positive ratio meets the requirement of P_{REQ}.

The values of m_i will be determined in the next subsection. If $m_i > 0$, multiple filtering vectors will be sent consecutively from the reader to tags in one round. If $m_i = 0$, no filtering vector is sent from the reader in this round. When this happens, it essentially allows multiple filtering vectors to be sent consecutively from tags to the reader (across multiple rounds). An illustration is given in Fig. 2.3.

2.3.6 Values of m_i

Let K be the total number of rounds. After all K rounds, we use X_{K+1} as our search result. There are in total K filtering vectors sent from tags to the reader. We know from Sect. 2.3.3 that each filtering vector can filter out half of non-members (in our case, tags in $X-W$). To meet the false-positive ratio requirement P_{REQ}, the following constraint should hold:

$$(P_{FV})^K \approx \left(\frac{1}{2}\right)^K \leq P_{REQ}. \tag{2.17}$$

Hence, the value of K is set to $\lceil -\frac{\ln P_{REQ}}{\ln 2} \rceil$. (We will discuss how to guarantee meeting the requirement P_{REQ} in Sect. 2.3.9.)

Next, we discuss how to set the values of m_i, $1 \leq i \leq K$, in order to minimize the execution time of each round. We use $FV(\cdot)$ to denote the filtering vector of a set. In phase one of the ith round, the reader builds m_i filtering vectors, denoted as

$FV_{i1}(X_i)$, $FV_{i2}(X_i)$, $\cdots$, $FV_{im_i}(X_i)$, which are consecutively broadcasted to the tags. From (2.12), we know the size of each filtering vector is $|X_i|/\ln 2$. After the filtration based on these vectors, the number of remaining candidate tags in Y_{i+1} is on average

$$\begin{aligned} |Y_{i+1}| &\approx |V_i| \times (P_{FV})^{m_i} + |W| \\ &\approx |V_i| \times (1/2)^{m_i} + |W| \\ &= |V_i|/2^{m_i} + |W|. \end{aligned} \tag{2.18}$$

In phase two of the ith round, the tags in Y_{i+1} use a time frame of $\frac{1}{\ln 2} \times |Y_{i+1}|$ slots to report their presence. After receiving the responses, the reader builds a filtering vector, denoted as $FV_i(Y_{i+1})$. After the filtration based on $FV_i(Y_{i+1})$, the size of the search result X_{i+1} is on average

$$\begin{aligned} |X_{i+1}| &\approx |U_i| \times P_{FV} + |W| \\ &\approx |U_i|/2 + |W| \\ &= (|X_i| + |W|)/2. \end{aligned} \tag{2.19}$$

We denote the transmission time of the ith round by $f(m_i)$. In order to make a fair comparison with CATS, we utilize the parameter setting that conforms with [28]. Therefore, $f(m_i) = \frac{1}{\ln 2} \times m_i \times |X_i| \times t_s + \frac{1}{\ln 2} \times |Y_{i+1}| \times t_s$, which is set to be:

$$f(m_i) = \frac{t_s}{\ln 2}\left(m_i|X_i| + |V_i|/2^{m_i} + |W|\right). \tag{2.20}$$

To find the value of m_i that minimizes $f(m_i)$, we take the first-order derivative and set the right side to zero.

$$\frac{df(m_i)}{dm_i} = \frac{t_s}{\ln 2}\left(|X_i| - \ln 2|V_i|/2^{m_i}\right) = 0 \tag{2.21}$$

Hence, the value of $f(m_i)$ is minimized when

$$m_i = \frac{\ln(\ln 2|V_i|/|X_i|)}{\ln 2}. \tag{2.22}$$

Because m_i cannot be a negative number, we reset $m_i = 0$ if $\frac{\ln(\ln 2|V_i|/|X_i|)}{\ln 2} < 0$. Furthermore, m_i must be an integer. If $\frac{\ln(\ln 2|V_i|/|X_i|)}{\ln 2}$ is not an integer, we round m_i either to the ceiling or to the floor, depending on which one results in a smaller value of $f(m_i)$.

For now, we assume that we know $|W|$ and $|Y|$ in our computation of m_i. Later we will show how to estimate these values on the fly in the execution of each round of our protocol. Initially, $|X_1|$ $(= |X|)$ is known. $|V_1|$ can be calculated from (2.16). Hence, the value of m_1 can be computed from (2.22). After that, we can estimate

$|Y_2|$, $|X_2|$, and $|V_2|$ based on (2.18), (2.19), and (2.16), respectively. From $|X_2|$ and $|V_2|$, we can calculate the value m_2. Following the same procedure, we can iteratively compute all values of m_i for $1 \leq i \leq K$.

We find it often happens that the m_i sequence has several consecutive zeros at the end, that is, $\exists p < K$, $m_i = 0$ for $i \in [p, K]$. In this case, we may be able to further optimize the value of m_p with a slight adjustment. We first explain the reason for $m_p = 0$: It costs some time for the reader to broadcast a filtering vector in phase one of the pth round. It is true that this filtering vector can reduce set Y_p, thereby reducing the frame size of phase two in the pth round. However, if the time cost of sending the filtering vector cannot be compensated by the time reduction of phase two, it will be better off to remove this filtering vector by setting $m_p = 0$. (This situation typically happens near the end of the m_i sequence because the number of unwanted tags in the remaining candidate set Y_p is already very small.) But if all values of m_i in the subsequent rounds (after m_p) are zeros, increasing m_p to a non-zero value m'_p may help reduce the transmission time of phase two of all subsequent rounds, and the total time reduction may compensate more than the time cost of sending those m'_p filtering vectors.

Consider the transmission time of these $(K - p + 1)$ rounds as a whole, denoted by $G(m'_p, p)$. It is easy to derive

$$G(m'_p, p) = \left(\frac{m'_p}{\ln 2} |X_p| + \frac{K - p + 1}{\ln 2} \left(\frac{|V_p|}{2^{m'_p}} + |W| \right) \right) t_s. \tag{2.23}$$

To minimize $G(m'_p, p)$, we have

$$m'_p = \begin{cases} 0 & \text{if } \gamma < 0 \\ \gamma & \text{if } \gamma \geq 0 \end{cases} \tag{2.24}$$

where $\gamma = \frac{\ln(\ln 2(K-p+1)|V_p|/|X_p|)}{\ln 2}$. As a result, m_p is updated to m'_p, while other m_i, $i \neq p$, remains unchanged.

Here, we give an example to illustrate how to calculate the values of m_i. Suppose $|X| = 5000$, $|Y| = 50{,}000$, $|W| = 500$, and $P_{REQ} = 0.001$, so $K = \lceil \frac{-\ln 0.001}{\ln 2} \rceil = 10$. Using (2.22), we can calculate the values from m_1 to m_{10}. The result is listed in Table 2.2. There is a sequence of zeros from m_7 to m_{10}. Thus, we can make an improvement using (2.24), and the optimized result is shown in Table 2.3.

Table 2.2 The initial values of m_i.

m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
3	1	0	1	0	1	0	0	0	0

Table 2.3 The optimized values of m_i.

m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
3	1	0	1	0	1	2	0	0	0

2.3.7 *Iterative Tag Search Protocol*

Having calculated the values of m_i, we can present our iterative tag search protocol (ITSP) based on the generalized approach in Sect. 2.3.5. The protocol consists of K iterative rounds. Each round consists of two phases. Consider the ith round, where $1 \leq i \leq K$.

2.3.7.1 Phase One

The RFID reader constructs m_i filtering vectors for X_i using m_i hash functions. According to (2.12), we set the size L_{X_i} of each filtering vector as

$$L_{X_i} = \frac{1}{\ln 2} \times |X_i|. \tag{2.25}$$

The RFID reader then broadcasts those filtering vectors one by one. Once receiving a filtering vector, each tag in Y_i maps its ID to a bit in the filtering vector using the same hash function that the reader uses to construct the filter. The tag checks whether this bit is "1". If so, it remains a candidate tag; otherwise, it is excluded as a non-candidate tag and drops out of the search process immediately. The set of remaining candidate tags is Y_{i+1}.

If the filtering vectors are too long, the reader divides each vector into blocks of a certain length (e.g., 96 bits) and transmits one block after another. Knowing which bit it is mapped to, each tag only needs to record one block that contains its bit.

From (2.13), we know that the false-positive probability after using m_i filtering vectors is $(P_{FV})^{m_i} \approx (1/2)^{m_i}$. Therefore, $|Y_{i+1}| \approx |V_i| \times (P_{FV})^{m_i} + |W| \approx |V_i|/2^{m_i} + |W|$.

2.3.7.2 Phase Two

The reader broadcasts the frame size $L_{Y_{i+1}}$ of phase two to the tags, where

$$L_{Y_{i+1}} = \frac{1}{\ln 2} \left(|V_i|/2^{m_i} + |W|\right). \tag{2.26}$$

After receiving $L_{Y_{i+1}}$, each tag in Y_{i+1} randomly maps its ID to a slot in the time frame using a hash function and transmits a one-bit short response to the reader in that slot. Based on the observed state (busy or empty) of the slots in the time frame, the reader builds a filtering vector, which is used to filter non-candidates from X_i.

The overall transmission time of all K rounds in the ITSP is

$$T_{ITSP} = \sum_{i=i}^{K} (m_i \times L_{X_i} + L_{Y_{i+1}}) \times t_s. \tag{2.27}$$

2.3.8 Cardinality Estimation

Recall from Sect. 2.3.6 that we must know the values of $|X_i|$, $|W|$, and $|V_i|$ to determine m_i, L_{X_i}, and $L_{Y_{i+1}}$. It is trivial to find the value of $|X_i|$ by counting the number of tags in the search result of the $(i-1)$th round. Meanwhile, we know $|V_i| \approx |V_{i-1}|/2^{m_{i-1}}$ and $|V_1| = |Y_1| - |W|$. Therefore, we only need to estimate $|W|$ and $|Y_1|$.

Besides serving as a filter, a filtering vector can also be used for cardinality estimation, a feature that is not exploited in [28]. Since no filtering vector is available at the very beginning, the first round of the ITSP should be treated separately: We may use the efficient cardinality estimation protocol ART [26] to estimate $|Y|$ (i.e., $|Y_1|$) if its value is not known at first. As for $|W|$, it is initially assumed to be $min\{|X|, |Y|\}$.

Next, we can take advantage of the filtering vector received by the reader in phase two of the ith ($i \geq 1$) round to estimate $|W|$ without any extra transmission expenditure. The estimation process is as follows: First, counting the actual number of "1" bits in the filtering vector, denoted as N_1^*, we know the actual false-positive probability of using this filtering vector, denoted by P_i^*, is

$$P_i^* = N_1^*/L_{Y_{i+1}}, \tag{2.28}$$

because an arbitrary unwanted tag has a chance of N_1^* out of $L_{Y_{i+1}}$ to be mapped to a "1" bit, where $L_{Y_{i+1}}$ is the size of the vector. Meanwhile, we can record the number of tags in the search results before and after the ith round, i.e., $|X_i|$ and $|X_{i+1}|$, respectively. We have $|X_i| = |U_i| + |W|$, $|X_{i+1}| = |U_{i+1}| + |W|$, and $|U_{i+1}| \approx |U_i| \times P_i^*$. Therefore,

$$|W| \approx \frac{|X_{i+1}| - |X_i| \times P_i^*}{1 - P_i^*}. \tag{2.29}$$

For the purpose of accuracy, we may estimate $|W|$ after every round, and obtain the average value.

2.3.9 Additional Filtering Vectors

Estimation may have error. Using the values of m_i and L_{Y_i} computed from estimated $|W|$ and $|Y_i|$, a direct consequence is that the actual false-positive ratio, denoted as P_T, can be greater than the requirement P_{REQ}. Fortunately, from (2.28), the reader is able to compute the actual false-positive ratio P_i^*, $1 \le i \le k$, of each filtering vector received in phase two of the ITSP. Thus, we have

$$P_T = \prod_{1}^{K} P_i^*. \tag{2.30}$$

If $P_T > P_{REQ}$, our protocol will automatically add additional filtering vectors to further filter X_{K+1} until $P_T \le P_{REQ}$ (as described in Sect. 2.3.4).

2.3.10 Hardware Requirement

ITSP cannot be supported by off-the-shelf tags that conform to the EPC Class-1 Gen-2 standard [9], whose limited hardware capability constrains the functions which can be supported. By our design, most of the ITSP protocol's complexity is on the reader side, but tags also need to provide certain hardware support. Besides the mandatory commands of C1G2 (e.g., Query, Select, and Read), in order for a tag to execute the ITSP protocol, we need a new command defined in the set of optional commands, asking each awake tag to listen to the reader's filtering vector, hash its ID to a certain slot of the vector for its bit value, keep silent and go sleep if the value is zero, and respond in a hashed slot (by making a transmission to make the channel busy) if the value is one. Note that the tag does not need to store the entire filtering vector, but instead only need to count to the slot it is hashed to, and retrieve the value (0/1) carried in that slot.

Hardware-efficient hash functions [1, 13, 22] can be found in the literature. A hash function may also be derived from the pseudo-random number generator required by the C1G2 standard. To keep the complexity of a tag's circuit low, we only use one uniform hash function $h(\cdot)$, and use it to simulate multiple independent hash functions: In phase one of the ith round, we use $h(\cdot)$ and m_i unique hash seeds $\{s_1, s_2, \cdots, s_{m_i}\}$ to achieve m_i independent hash outputs. Thus, a tag id is mapped to bit locations $(h(id \oplus s_1) \bmod L_{X_i})$, $(h(id \oplus s_2) \bmod L_{X_i})$, $\cdots$, $(h(id \oplus s_{m_i}) \bmod L_{X_i})$ in the m_i filtering vectors, respectively. Each hash seed, together with its corresponding filtering vector, will be broadcast to the tags. In phase two of the ith round, the reader generates a new hash seed s' and sends it to the tags. Each candidate tag in Y_{i+1} maps its id to the slot of index $\big(h(id \oplus s') \bmod L_{Y_{i+1}}\big)$, and then transmits a one-bit short response to the reader in that slot.

2.4 ITSP over Noisy Channel

So far the ITSP assumes that the wireless channel between the RFID reader and tags is reliable. Note that the CATS protocol does not consider channel error, either. However, it is common in practice that the wireless channel is far from perfect due to many different reasons, among which interference noise from nearby equipment, such as motors, conveyors, robots, wireless LAN's, and cordless phones, is a crucial one. Therefore, this section is to enhance ITSP by making it robust against noise interference.

2.4.1 ITSP with Noise on Forward Link

The reader transmits at a power level much higher than the tags (which after all backscatter the reader's signals in the case of passive tags). It has been shown that the reader may transmit more than one million times higher than tag backscatter [14]. Hence, the forward link (reader to tag) communication is more resilient against channel noise than the reverse link (tag to reader). To provide additional assurance against noise for forward link, we may use CRC code for error detection. The C1G2 standard requires the tags to support the computation of CRC-16 (16-bit CRC) [9], which therefore can also be adopted by future tags modified for ITSP. Each filtering vector built by the reader can be regarded as a combination of many small segments with fixed size of l_S bits (e.g., $l_S = 80$). For each segment, the reader computes its 16-bit CRC and appends it to end of that segment. Those segments are then concatenated and transmitted to tags. When a tag receives a filtering vector, it first finds the segment it hashes to and computes the CRC of that segment. If the calculated CRC matches the attached one, it will determine its candidacy by checking the bit in the segment which it maps to. For mismatching CRC, the tag knows that the segment has been corrupted, and it will remain as a candidate tag regardless of the value of the bit which it maps to.

Suppose we let $l_S = 80$, then

$$L_{X_i} = \frac{\frac{1}{\ln 2} \times |X_i|}{l_S} \times (l_S + 16) = \frac{1.2|X|}{\ln 2}. \tag{2.31}$$

We assume the probability that the noise corrupts each segment is P_S (P_S is expected to be very small as explained above). A corrupted segment can be thought as consisting of all "1"s. Hence, the false-positive probability for a filtering vector sent by reader, denoted by P_{RT}, is roughly

$$\begin{aligned} P_{RT} &\approx \frac{\frac{L_{X_i}}{96} \times P_S \times l_S + \frac{L_{X_i}}{96} \times (1 - P_S) \times l_S \times P_{FV}}{\frac{L_{X_i}}{96} \times l_S} \\ &= \frac{1 + P_S}{2}. \end{aligned} \tag{2.32}$$

We can also get

$$|\, Y_{i+1}| \approx |V_i| \times (P_{RT})^{m_i} + |W| \tag{2.33}$$

and now (2.20) can be rewritten as

$$f(m_i) = \frac{t_s}{\ln 2}\left(1.2m_i|X_i| + \left(\frac{1 + P_{RT}}{2}\right)^{m_i} |V_i| + |W|\right). \tag{2.34}$$

Therefore, $f(m_i)$ is optimized when

$$m_i = \frac{\ln[(\ln 2 - \ln(1 + P_{RT}))|V_i|/1.2|X_i|]}{\ln 2 - \ln(1 + P_{RT})}. \tag{2.35}$$

2.4.2 ITSP with Noise on Reverse Link

Now let us study the noise on the reverse link and its effect on the ITSP. Since the backscatter from a tag is much weaker than the signal transmitted by the reader, the reverse link is more likely to be impacted by noise.

First, channel noise may corrupt a would-be empty slot into a busy slot. The original empty slot is supposed to be translated into a "0" bit in the filtering vector by the reader; if a candidate tag is mapped to that bit, it is ruled out immediately. However, if that slot is corrupted and becomes a busy slot, the corresponding bit turns into "1"; a tag mapped to that bit will remain a candidate tag, thereby increasing the false-positive probability of the filtering vector.

Second, noise may also occur during a busy slot. Although the noise and the transmissions from tags may partially cancel each other in a slot if they happen to reach the reader in opposite phase, it is extremely unlikely that they will exactly eliminate each other. As long as the reader can still detect some energy, regardless of its source (it may even come from the noise), that slot will be correctly determined as a busy slot, and the corresponding bit in the filtering vector is set to "1" just as it is supposed to be. However, if we take the propagation path loss, including reflection loss, attenuation loss, and spreading loss [11], into account, there is still a chance that a busy slot may not be detected by the reader. This may happen in a time-varying channel where the reader may fail in receiving a tag's signal during a deeply faded slot when the tag transmits. We stress that this is not a problem unique to ITSP, but all protocols that require communications from tags to readers will suffer from this problem if it happens that the reader cannot hear the tags. ITSP is not robust against this type of error. But there exists ways to alleviate this problem—for instance, each filtering vector from tags to the reader is transmitted twice. As long as a slot is busy in one of two transmissions, the slot is considered to be busy.

Next, we will investigate the reverse link with noise interference for ITSP under two error models.

2.4.2.1 ITSP Under Random Error Model (ITSP-rem)

The random error model is characterized by a parameter called error rate P_{ERR}, which means every slot independently has a probability P_{ERR} to be corrupted by the noise. Influencing by the channel noise, the reader can detect more busy slots as some empty slots turn into busy ones, which raises the false-positive probability of phase-two filtering vectors. Suppose the frame size of phase two in a certain round is l, the original number of busy slots is about $l \times P_{FV} \approx l/2$. At the reader's side, however, the number of busy slots averagely increases to $l/2 + l/2 \times P_{ERR} = \frac{(1+P_{ERR})\times l}{2}$. After encoding the slot status into a filtering vector, the false-positive probability of that filtering vector is

$$P'_{FV} \approx \frac{\frac{(1+P_{ERR})\times l}{2}}{l} = \frac{1+P_{ERR}}{2}. \tag{2.36}$$

To satisfy the false-positive ratio requirement, $\left(P'_{FV}\right)^K \leq P_{REQ}$ should hold. Therefore, the search process of ITSP-rem contains at least

$$K = \lceil \frac{\ln P_{REQ}}{\ln[(1+P_{ERR})/2]} \rceil \tag{2.37}$$

rounds. Also, we can derive

$$\begin{aligned} |X_{i+1}| &\approx |U_i| \times P'_{FV} + |W| \\ &\approx |U_i|(1+P_{ERR})/2 + |W|. \end{aligned} \tag{2.38}$$

With K, $|X_i|$, $|Y_i|$ and m_i, $1 \leq i \leq K$, the search time of ITSP-rem can be calculated using (2.31) (2.26) (2.27).

2.4.2.2 ITSP Under Burst Error Model (ITSP-bem)

In telecommunication, a burst error is defined as a consecutive sequence of received symbols, where the first and last symbols are in error, and there exists no continuous subsequence of m (m is a specified parameter called the guard band of the error burst) correctly received symbols within the error burst [10]. A burst error model describes the number of bursts during an interval and the number of incorrect symbols in each burst error, which differs greatly from the random error model.

According to the burst error model presented in [6], both the number of bursts in an interval and the number of errors in each burst have Poisson distributions. Assume the expected number of bursts in an l-bit interval is η, the probability distribution function for the number of bursts can be expressed as

$$h(x) = \sum_{i=0}^{\infty} \frac{\eta^i}{i!} e^{-\eta} \delta_{xi}, \tag{2.39}$$

where δ_{xi} is the Kronecker delta function [18]. Meanwhile, if the mean value of errors due to a burst in the l bits is τ, then the probability distribution function of the number of error is given by

$$g(y) = \sum_{j=0}^{\infty} \frac{\tau^j}{j!} e^{-\tau} \delta_{yj}. \tag{2.40}$$

Therefore, the probability of having w errors in an interval of l bits is

$$P_l(w) = e^{-\eta} \frac{\tau^w}{w!} \sum_{i=0}^{\infty} \frac{i^w}{i!} \eta^i e^{-i\tau}. \tag{2.41}$$

In other words, for a frame with l slots, the probability that w slots will be corrupted by the burst noise is $P_l(w)$.

Now we evaluate the ITSP under the burst error model, denoted as ITSP-bem. Given a filtering vector with size of l-bit, recall from (2.41) that the probability of having w errors in this l-bit vector is $P_l(w)$. In this case, each original “0” bit has a probability $\frac{w}{l}$ to be corrupted by the errors, and becomes a “1” bit. Consequently, the false-positive probability of the filtering vector is expected to be:

$$P'_{FV} \approx \frac{1}{2} + \frac{1}{2} \sum_{w=0}^{l} P_l(w) \times \frac{w}{l}. \tag{2.42}$$

After obtaining the value of P'_{FV}, the ITSP-bem can use (2.37), (2.38), to determine the values of other necessary parameters.

2.5 Performance Evaluation

2.5.1 Performance Metric

We compare our protocol ITSP with CATS [28], the polling protocol (Sect. 2.2.2), the optimal DFSA (dynamic frame slotted ALOHA), and a tag identification protocol with collision recovery [15], denoted as CR, which identifies 4.8 tags per slot on average, about 13 times the speed of the optimal DFSA. For ITSP and CATS, their Bloom filters (or filtering vectors) constitute most of the overall transmission overhead, while other transmission cost, such as transmission of hash seeds, is comparatively negligible. Both protocols need to estimate the number of tags in the system, $|Y|$, as a pre-protocol step. According to the results presented in [28], the time for estimating $|Y|$ takes up less than 2 % of the total execution time of CATS. Hence, we do not count the estimation time of $|Y|$ in the simulation results because it is relatively small and does not affect fair comparison as both protocols need it. Consequently, the key metric concerning the time efficiency is the total size of Bloom filters or filtering vectors, and then (2.8) can be used for calculating the search time required by CATS, while (2.27) for ITSP.

After the search process is completed, we will calculate the false -positive ratio P_{FP} using $P_{FP} = \frac{|W^* - W|}{|X - W|}$, where W^* is the set of tags in the search result and W is the actual set of wanted tags in the coverage area. P_{FP} will be compared with P_{REQ} to see whether the search result meets the false -positive ratio requirement.

2.5.2 Performance Comparison

We evaluate the performance of our protocol and compare it with the CATS protocol. In the first set of simulations, we set $P_{REQ} = 0.001$, fix $|Y| = 50{,}000$, vary $|X|$ from 5000 to 640,000, and let $R_{INTS} = 0.1, 0.3, 0.5, 0.7, 0.9$. In the second set of simulations, we set $P_{REQ} = 0.001$, fix $|X| = 10{,}000$, vary $|Y|$ from 1250 to 40,000 to investigate the scalability of ITSP with tag population from a large range, and let $R_{INTS} = 0.1, 0.3, 0.5, 0.7, 0.9$. For simplicity, we assume $t_{id} = 96t_s$, and $t_l = 137t_s$, in which a 9-bit QueryAdjust or a 4-bit QueryRep command, a 96-bit ID and two 16-bit random numbers can be transmitted. Tables 2.4 and 2.5 show the number of t_s slots needed by the protocols under different parameter settings. Each data point in these tables or other figures/tables in the rest of the section is the average of 500 independent simulation runs with ±5 % or less error at 95 % confidence level.

From the tables, we observe that when R_{INTS} is small (which means $|W|$ is small), the ITSP performs much better than the CATS protocol. For example, in Table 2.4, when $R_{INTS} = 0.1$, the ITSP reduces the search time of the CATS protocol by as much as 90.0 %. As we increase R_{INTS} (which implies larger $|W|$), the gap between the performance of the ITSP and the performance of the CATS gradually shrinks.

Table 2.4 Performance comparison of tag search protocols. CR means a tag identification protocol with collision recovery techniques. $|Y| = 50{,}000$, $P_{REQ} = 0.001$

	ITSP (R_{INTS})							
$\|X\|$	0.1	0.3	0.5	0.7	0.9	CATS	Polling	CR
5,000	61,463	96,989	105,828	108,346	124,553	126,370	485,000	1,427,083
10,000	108,017	145,553	206,709	199,586	231,236	238,313	970,000	1,427,083
20,000	185,204	255,898	335,426	397,462	403,954	447,772	1,940,000	1,427,083
40,000	304,767	467,433	512,156	598,718	678,066	837,837	3,880,000	1,427,083
80,000	414,686	590,150	656,426	721,347	721,347	1,560,259	7,760,000	1,427,083
160,000	472,677	630,669	721,347	721,347	721,347	2,889,689	15,520,000	1,427,083
320,000	529,835	668,794	721,347	721,347	721,347	5,317,715	31,040,000	1,427,083
640,000	573,270	696,015	721,347	721,347	721,347	10,533,732	62,080,000	1,427,083

Table 2.5 Performance comparison of tag search protocols. CR means a tag identification protocol with collision recovery techniques. $|X| = 10{,}000$, $P_{REQ} = 0.001$

	ITSP (R_{INTS})							
$\|Y\|$	0.1	0.3	0.5	0.7	0.9	CATS	Polling	CR
1,250	13,047	17,364	18,033	18,033	18,033	164,589	970,000	35,677
2,500	24,289	33,337	36,067	36,067	36,067	175,960	970,000	71,354
5,000	42,835	62,862	68,528	72,134	72,134	190,387	970,000	142,708
10,000	73,909	109,281	119,022	137,056	144,269	204,814	970,000	285,417
20,000	95,833	132,546	169,065	167,713	192,960	219,241	970,000	570,833
40,000	111,904	152,606	174,926	228,215	232,904	233,668	970,000	1,141,667

In particular, the CATS performs poorly when $|X| \geq |Y|$. But the ITSP can work efficiently in all cases. In addition, the ITSP is also much more efficient than the polling protocol, and any tag identification protocol with/without CR techniques. Even in the worst case, the ITSP only takes about half of the execution time of a tag identification protocol with CR techniques. (Note that the identification process actually takes much more time since the throughput 4.8 tags per slot may not be achievable in practical and the duration of each slot is longer.) In practice, the wanted tags may be spatially distributed in many different RFID systems (e.g., warehouses in the example we use in the introduction), and thus R_{INTS} can be small. The ITSP is a much better protocol for solving the tag search problem in these practical scenarios.

Another performance issue we want to investigate is the relationship between the search time and P_{REQ}. The polling protocol, DFSA, and CR do not have false positive. Our focus will be on ITSP and CATS. We set $|X| = 5000$, 20,000 or 80,000, $|Y| = 50{,}000$, vary R_{INTS} from 0.1 to 0.9, and vary P_{REQ} from 10^{-6} to 10^{-2}. Figure 2.4 compares the search times required by the CATS and the ITSP under different false -positive ratio requirements. Generally speaking, the gap between the search time required by the ITSP and the search time by the CATS keeps getting larger with the decrease of P_{REQ}, particularly when R_{INTS} is small. For example, in

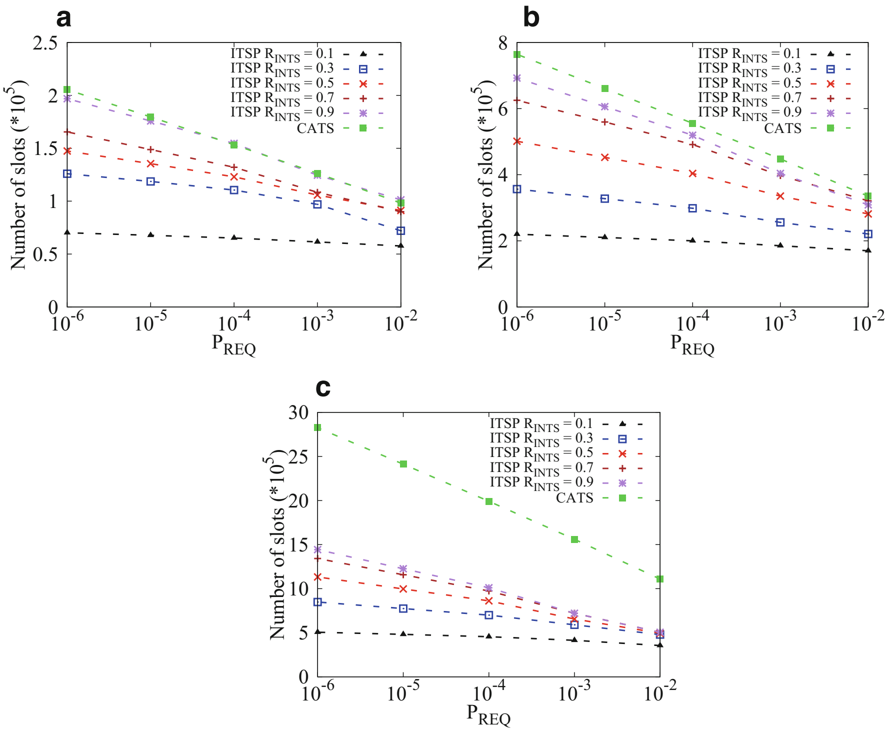


Fig. 2.4 Relationship between search time and P_{REQ}. Parameter setting: $|Y| = 50{,}000$; (**a**) $|X| = 5000$, (**b**) $|X| = 20{,}000$, (**c**) $|X| = 80{,}000$

Fig. 2.4c, when $P_{REQ} = 10^{-2}$ and $R_{INTS} = 0.1$, the search time by the ITSP is about one third of the time by the CATS; when we reduce P_{REQ} to 10^{-6}, the time by the ITSP becomes about one fifth of the time by the CATS. The reason is as follows: When R_{INTS} is small, $|W|$ is small and most tags in X and Y are non-candidates. After several ITSP rounds, as many non-candidates are filtered out iteratively, the size of filtering vectors decreases exponentially and therefore subsequent ITSP rounds do not cause much extra time cost. This merit makes the ITSP particularly applicable in cases where the false -positive ratio requirement is very strict, requiring many ITSP rounds. On the contrary, the CATS protocol does not have this capability of exploiting low R_{INTS} values.

2.5.3 False -Positive Ratio

Next, we examine whether the search results after execution of the ITSP will indeed meet the requirement of P_{REQ}. In this simulation, we set the false-positive ratio requirement based on the following formula:

$$P_{REQ} \leq \frac{|W|}{\lambda\,(|X| - |W|)}, \tag{2.43}$$

where λ is a constant. We use an example to give the rationale: Consider an RFID system with $|X| = 20{,}000$. If $|W| = 10{,}000$, $P_{REQ} = 0.01$ may be good enough because the number of false positives is about $(|X| - |W|) \times P_{REQ} = 100$, which is much fewer than $|W|$. However, if $|W| = 10$, $P_{REQ} = 0.01$ may become unacceptable since $(|X| - |W|) \times P_{REQ} \approx 200 \gg |W|$. Therefore, it is desirable to set the value of P_{REQ} such that the number of false positives in the search result is much smaller than $|W|$, namely $(|X| - |W|) \times P_{REQ} \leq \frac{1}{\lambda}|W|$. Let $\lambda = 10$ and we test the ITSP under three different parameter settings:

(1) $|X| = 5000$, $|Y| = 50{,}000$, and R_{INTS} varies from 0.1 to 0.9, i.e., $|W|$ varies from 500 to 4500. $P_{REQ} \leq \frac{500}{10\times(5000-500)} \approx 0.01111$. We set $P_{REQ} = 10^{-2}$.
(2) $|X| = 20{,}000$, $|Y| = 50{,}000$, and R_{INTS} varies from 0.01 to 0.9, i.e., $|W|$ varies from 200 to 18,000. $P_{REQ} \leq \frac{200}{10\times(20{,}000-200)} \approx 0.00101$. We set $P_{REQ} = 10^{-3}$.
(3) $|X| = 80{,}000$, $|Y| = 50{,}000$, and R_{INTS} varies from 0.01 to 0.9, i.e., $|W|$ varies from 500 to 45,000. $P_{REQ} \leq \frac{500}{10\times(80{,}000-500)} \approx 0.00063$. We set $P_{REQ} = 10^{-4}$.

For each parameter setting, we repeat the simulation 500 times to obtain the average false -positive ratio.

Figure 2.5 shows the simulation results. In (a), (b), and (c), we can see that the average P_{FP} is always smaller than the corresponding P_{REQ}. Hence, the search results using the ITSP meet the prescribed requirement of false -positive ratio in the average sense.

If we look into the details of individual simulations, we find that a small fraction of simulation runs have P_{FP} beyond P_{REQ}. For example, Fig. 2.6 depicts the results of 500 runs with $|X| = 5000$, $|Y| = 50{,}000$, $|W| = 500$, and $P_{REQ} = 10^{-2}$. There are about 5 % runs having $P_{FP} > P_{REQ}$, but that does not come as a surprise because the false -positive ratio in the context of filtering vectors (ITSP) or Bloom filters (CATS) is defined in a probability way: The probability for each tag in $X - W$ to be misclassified as one in W is no greater than P_{REQ}. This probabilistic definition enforces a requirement P_{REQ} in an average sense, but not absolutely for each individual run.

2.5.4 Performance Evaluation Under Channel Error

2.5.4.1 Performance of ITSP-rem and ITSP-bem

We evaluate the performance of ITSP-rem and ITSP-bem. To simulate the error rate P_{ERR} in ITSP-rem, we employ a pseudo-random number generator, which generates random real numbers uniformly in the range [0, 1]. If a bit in the filtering vector is "0" and the generated random number is in [0, P_{ERR}], that bit is flipped to "1". P_S can be simulated in a similar way. As for the burst error in ITSP-bem, we first

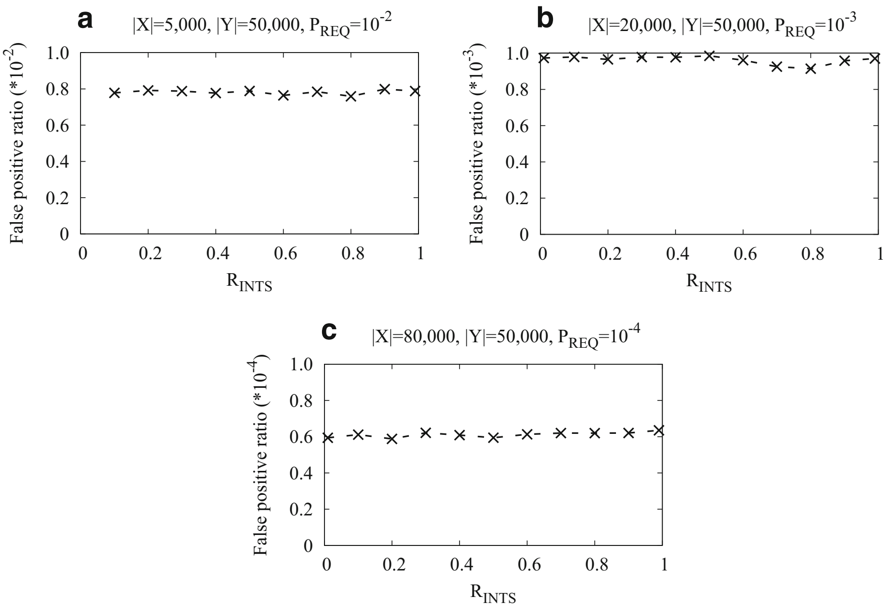


Fig. 2.5 False -positive ratio after running the ITSP

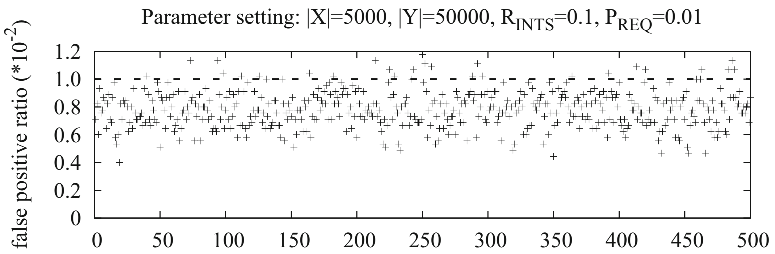


Fig. 2.6 False -positive ratio by the ITSP of 500 runs

calculate the values of $P_l(w)$ with different w for a given l. Then each w is assigned with a non-overlapping range in [0, 1], whose length is equal to the value of $P_l(w)$. For each interval, we generate a random number and check which range the number locates, thereby determining the number of errors in that interval.

We set $P_{REQ} = 0.001$, $P_S = 0.01$, and $R_{INTS} = 0.1, 0.5, 0.9$, respectively. The values of $|X|$ and $|Y|$ are the same as those in Tables 2.4 and 2.5. l_s is set to 80 bits and a 16-bit CRC is appended to each segment on forward link for integrity check. For ITSP-rem, we consider two cases with $P_{ERR} = 5\,\%$ and $10\,\%$, respectively. For ITSP-bem, the prescribed parameters are set to be: $\eta = 0.135$, $\tau = 7.10$ with each interval to be 96 bits [6].

Table 2.6 Performance comparison. $|Y| = 50{,}000$, $R_{INTS} = 0.1$, $P_{REQ} = 0.001$

$\|X\|$	ITSP	ITSP-rem		ITSP-bem
		$P_{ERR} = 5\,\%$	$P_{ERR} = 10\,\%$	
5,000	61,463	74,288	75,812	72,144
10,000	108,017	129,995	133,022	125,779
20,000	185,204	241,026	247,824	238,962
40,000	304,767	361,242	398,198	358,361
80,000	414,686	441,365	458,433	437,256
160,000	472,677	504,565	545,338	499,058
320,000	529,835	567,403	630,174	560,456
640,000	573,270	626,379	690,400	618,913

Table 2.7 Performance comparison. $|Y|$=50,000, R_{INTS}=0.5, $P_{REQ} = 0.001$

$\|X\|$	ITSP	ITSP-rem		ITSP-bem
		$P_{ERR} = 5\,\%$	$P_{ERR} = 10\,\%$	
5,000	105,828	160,481	166,469	153,838
10,000	206,709	211,513	221,771	210,805
20,000	335,426	371,974	391,983	370,557
40,000	512,156	577,305	617,196	577,305
80,000	656,426	735,592	789,874	735,592
160,000	721,347	793,482	865,617	793,482
320,000	721,347	793,482	865,617	793,482
640,000	721,347	793,482	865,617	793,482

Tables 2.6, 2.7, 2.8, 2.9, 2.10, and 2.11 show the number of t_s slots needed under each parameter setting. The second column presents the results of ITSP when the channel is perfectly reliable. The third and fourth columns present the results of ITSP-rem with an error rate of 5 % or 10 %. The fifth column presents the results of ITSP-bem. It is not surprising that the search process under noisy channel generally takes more time due to the use of CRC and the higher false-positive probability of filtering vectors, and the execution time of the ITSP-rem is usually longer in a channel with a higher error rate. An important positive observation is that the performance of ITSP gracefully degrades in all simulations. The increase in execution time for both ITSP-rem and ITSP-bem is modest, compared to ITSP with a perfect channel. For example, even when the error rate is 10 %, the execution time of ITSP-rem is about 10–30 % higher than that of ITSP. This modest increase demonstrates the practicality of our protocol under noisy channel.

2.5.4.2 False-Positive Ratio of ITSP-rem and ITSP-bem

We use the same parameter settings in Sect. 2.5.3 to examine the accuracy of search results by ITSP-rem and ITSP-bem. Meanwhile, for ITSP-rem, we set $P_{ERR} = 5\,\%$ or $10\,\%$. For ITSP-bem, the required input parameter setting is $\eta = 0.135$ and $\tau = 7.10$, with each 96-bit interval. Simulation results are delineated in Fig. 2.7,

Table 2.8 Performance comparison. $|Y|$=50,000, R_{INTS}=0.9, $P_{REQ} = 0.001$

$\|X\|$	ITSP	ITSP-rem		ITSP-bem
		$P_{ERR} = 5\,\%$	$P_{ERR} = 10\,\%$	
5,000	124,553	156,041	163,718	155,972
10,000	231,236	275,394	290,493	275,256
20,000	403,954	454,929	486,150	454,929
40,000	678,066	752,753	814,890	752,753
80,000	721,347	793,482	865,617	793,482
160,000	721,347	793,482	865,617	793,482
320,000	721,347	793,482	865,617	793,482
640,000	721,347	793,482	865,617	793,482

Table 2.9 Performance comparison. $|X|$=10,000, R_{INTS}=0.1, $P_{REQ} = 0.001$

$\|Y\|$	ITSP	ITSP-rem		ITSP-bem
		$P_{ERR} = 5\,\%$	$P_{ERR} = 10\,\%$	
1,250	13,047	14,868	15,898	14,174
2,500	24,289	26,626	28,617	25,283
5,000	42,835	46,994	50,863	44,393
10,000	73,909	76,807	84,135	75,983
20,000	95,833	103,255	106,693	102,121
40,000	111,904	133,043	137,348	130,382

Table 2.10 Performance comparison. $|X|$=10,000, R_{INTS}=0.5, $P_{REQ} = 0.001$

$\|Y\|$	ITSP	ITSP-rem		ITSP-bem
		$P_{ERR} = 5\,\%$	$P_{ERR} = 10\,\%$	
1,250	18,033	19,837	21,640	19,837
2,500	36,067	39,674	43,280	39,674
5,000	68,528	77,021	82,448	77,021
10,000	119,022	134,208	143,261	134,208
20,000	169,065	202,891	212,105	202,467
40,000	174,926	214,563	224,227	213,970

Table 2.11 Performance comparison. $|X|$=10,000, R_{INTS}=0.9, $P_{REQ} = 0.001$

$\|Y\|$	ITSP	ITSP-rem		ITSP-bem
		$P_{ERR} = 5\,\%$	$P_{ERR} = 10\,\%$	
1,250	18,033	19,837	21,640	19,837
2,500	36,067	39,674	43,280	39,674
5,000	72,134	79,348	86,561	79,348
10,000	144,269	158,696	173,123	158,696
20,000	192,960	217,245	232,272	217,245
40,000	232,904	261,277	277,300	261,173

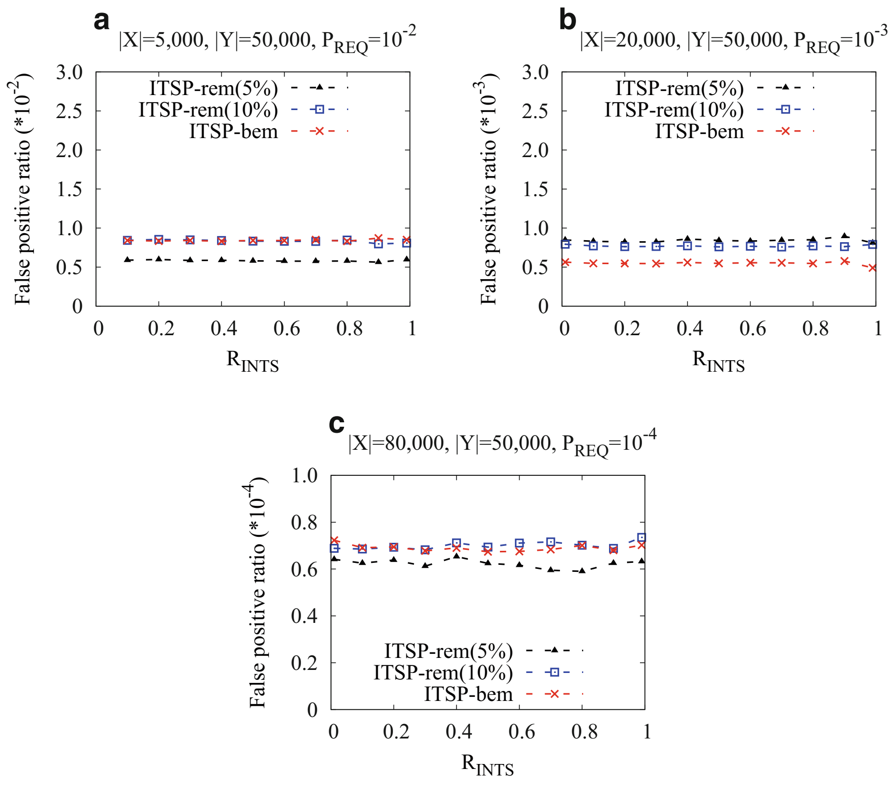


Fig. 2.7 False -positive ratio after running ITSP-rem, ITSP-bem, and CATS. (**a**) $|X|$=5000, $|Y|$=50,000, $P_{REQ} = 10^{-2}$, (**b**) $|X|$=20,000, $|Y|$=50,000, $P_{REQ} = 10^{-3}$, (**c**) $|X|$=80,000, $|Y|$=50,000, $P_{REQ} = 10^{-4}$

where the error rate is given between the parentheses after ITSP-bem. Clearly, the false-positive ratio in the search results after executing ITSP-rem or ITSP-bem is always within the bound of P_{REQ}. These results confirm that the false-positive ratio requirement is met under noisy channel.

2.5.4.3 Signal Loss Due to Fading Channel

We consider the scenario of a time-varying channel in which it may happen that a signal from a tag is not received by the reader in a deep fading slot. Although we consider this condition is relatively rare in an RFID system that is configured to work stably, we acknowledge in Sect. 2.4.2 that ITSP (or CATS) is not robust against this type of error. However, the problem can be alleviated by the tags transmitting each filtering vector twice. Figure 2.8 shows the simulation results under parameters $|X| = 10000$, $|Y| = 5000$, $|W| = 500$, and $P_{REQ} = 0.01$. The horizontal axis

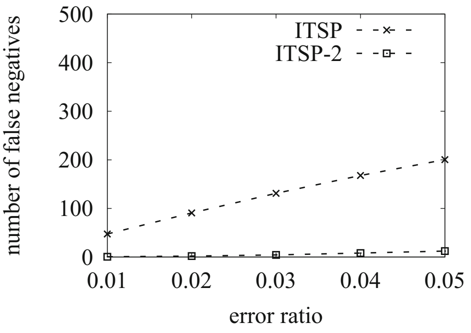


Fig. 2.8 False negatives due to signal loss in time-varying channel

shows the error rate, which is defined as the fraction of slots in deep fading, causing complete signal loss. ITSP-2 denotes the approach of transmitting each filtering vector from tags to the reader twice. When a wanted tag in W is not identified, we call it a false negative. The simulation results show that ITSP incurs significant false negatives when the error rate becomes large. For example, when the error rate is 2 %, the average number of false negatives is 90.7. ITSP-2 works very well in reducing this number. When the error rate is 2 %, its number of false negatives is just 1.95.

2.6 Summary

This chapter discusses the tag search problem in large-scale RFID systems. We present an iterative tag search protocol (ITSP) that improves time efficiency and eliminates the limitation of prior solutions. Moreover, we extend the ITSP to work under noisy channel. The main contributions of our work are summarized as follows: (1) The iterative method of ITSP based on filtering vectors is very effective in reducing the amount of information to be exchanged between tags and the reader, and consequently saves time in the search process; (2) the ITSP performs much better than the existing solutions; (3) the ITSP works well under all system conditions, particularly in situations of $|X| \gg |Y|$ when CATS works poorly; (4) the ITSP is improved to work effectively under noisy channel.

References

1. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash functions and RFID tags: mind the gap. In: Proceedings of CHES, pp. 283–299 (2008)
2. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: a survey. Internet Math. **1**(4), 485–509 (2003)
3. Castiglione, P., Ricciato, F., Popovski, P.: Pseudo-random Aloha for inter-frame soft combining in RFID systems. In: Proceedings of IEEE DSP, pp. 1–6 (2013)

4. Cha, J.R., Kim, J.H.: Dynamic framed slotted ALOHA algorithms using fast tag estimation method for RFID systems. In: Proceedings of IEEE Consumer Communications and Networking Conference (CCNC) (2006)
5. Choi, J., Lee, C.: A cross-layer optimization for a LP-based multi-reader coordination in RFID systems. In: Proceedings of IEEE GLOBECOM, pp. 1–5 (2010)
6. Cornaglia, B., Spini, M.: New statistical model for burst error distribution. Eur. Trans. Telecommun. **7**, 267–272 (1996)
7. Dan, L., Wei, P., Wang, J., Tan, J.: TFDMA: a scheme to the RFID reader collision problem based on graph coloration. In: Proceedings of IEEE SOLI, pp. 502–507 (2008)
8. Eom, J., Lee, T.: Accurate tag estimation for dynamic framed-slotted ALOHA in RFID systems. In: Proceedings of IEEE Communication Letters, pp. 60–62 (2010)
9. EPC Radio-Frequency Identity Protocols Class-1 Gen-2 UHF RFID Protocol for Communications at 860MHz-960MHz, EPCglobal. Available at http://www.epcglobalinc.org/uhfclg2 (2011)
10. Federal Standard 1037C. Available at http://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm (1996)
11. Fletcher, R., Marti, U.P., Redemske, R.: Study of UHF RFID signal propagation through complex media. In: IEEE Antennas and Propagation Society International Symposium, vol. 1B, pp. 747–750 (2005)
12. Fyhn, K., Jacobsen, R.M., Popovski, P., Scaglione, A., Larsen, T.: Multipacket reception of passive UHF RFID tags: a communication theoretic approach. IEEE Trans. Signal Process. **59**(9), 4225–4237 (2011)
13. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight Hash functions. In: Proceedings of CRYPTO, pp. 222–239 (2011)
14. NIST: RFID Communication and Interference. White paper, Grand Prix Application Series (2007)
15. Kaitovic, J., Rupp, M.: Improved physical layer collision recovery receivers for RFID readers. In: Proceedings of IEEE RFID, pp. 103–109 (2014)
16. Kaitovic, J., Langwieser, R., Rupp, M.: A smart collision recovery receiver for RFIDs. EURASIP J. Embed. Syst. **2013**, 1–19 (2013)
17. Kang, Y., Kim, M., Lee, H.: A hierarchical structure based reader anti-collision protocol for dense RFID reader networks. In: Proceedings of ICACT, pp. 164–167 (2011)
18. Kronecker delta. Available at http://en.wikipedia.org/wiki/Kronecker_delta
19. Lee, S., Joo, S., Lee, C.: An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification. In: Proceedings of IEEE MobiQuitous (2005)
20. Nguyen, C.T., Hayashi, K., Kaneko, M., Popovski, P., Sakai, H.: Probabilistic dynamic framed slotted ALOHA for RFID tag identification. Wirel. Pers. Commun. **71**, 2947–2963 (2013)
21. Onat, I., Miri, A.: A tag count estimation algorithm for dynamic framed ALOHA based RFID MAC protocols. In: Proceedings of IEEE ICC, pp. 1–5 (2011)
22. O'Neill, M.: Low-cost SHA-1 hash function architecture for RFID tags. In: Proceedings of RFIDSec (2008)
23. Qiao, Y., Li, T., Chen, S.: One memory access bloom filters and their generalization. In: Proceedings of IEEE INFOCOM, pp. 1745–1753 (2011)
24. Ricciato, F., Castiglione, P.: Pseudo-random ALOHA for enhanced collision-recovery in RFID. IEEE Commun. Lett. **17**(3), 608–611 (2013)
25. Schoute, F.C.: Dynamic frame length ALOHA. IEEE Trans. Commun. **31**, 565–568 (1983)
26. Shahzad, M., Liu, A.: Every bit counts - fast and scalable RFID estimation. In: Proceedings of ACM Mobicom (2012)
27. Stefanovic, C., Popovski, P.: ALOHA random access that operates as a rateless code. IEEE Trans. Commun. **61**(11), 4653–4662 (2013)
28. Zheng, Y., Li, M.: Fast tag searching protocol for large-scale RFID systems. IEEE/ACM Trans. Networking **21**(3), 924–934 (2012)