# Efficient Information Collection Protocols for Sensor-augmented RFID Networks

Shigang Chen †  Ming Zhang†  Bin Xiao‡
† Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL, USA
‡ Hong Kong Polytechnic University, Hong Kong, China

***Abstract*—Similar to the revolutionary change that the barcode system brought to the retail industry, the RFID technologies are expected to revolutionize the warehouse and inventory management. After RFID tags are deployed to make the attached objects wirelessly identifiable, a natural next step is to invent new ways to benefit from this "infrastructure". For example, sensors may be added to these tags to gather real-time information about the state of the objects or about the environment where these objects reside. This leads to the problem of designing efficient protocols to collect such information from the tags. It is a new problem that the existing work cannot solve well. In this paper, we first show that a straightforward polling solution will not be efficient. We then propose a single-hash information collection protocol that works much better than the polling solution. However, a wide gap still exists between the execution time of this protocol and a lower bound that we establish. Finally, we propose a multi-hash information collection protocol that further reduces the expected execution time to within 1.61 times the lower bound.**

## I. Introduction

The barcode system brought a revolutionary change in the retail industry. Information can be embedded in the barcode. In particular, a product ID can be encoded. Once a reader retrieves the ID, it can use the ID to search a database to find all information about the product, which may include price, features, or even manufacture and shipping history. However, barcode has a very small reading range. This is fine when used for checkout in a retail store, but it is not suitable for warehouse management. RFID technologies overcome this deficiency. A reader can wirelessly access the ID of a RFID tag from a distance even without line of sight. The most popular tags used in today's industry are passive ones. They do not carry batteries, but instead rely on radio waves emitted by the reader for energy to power its circuit and transmit information back to the reader.

For warehouse management and inventory control, it is certainly much easier to use passive tags than barcodes. A person can carry a mobile reader and walk through the warehouse to collect information from the tags without having to move close to each individual object. These tags are cheap, but their operational range is short, especially in an indoor environment with lots of racks and mechanise. They do not have much memory space or processing capability for sophisticated functions. If the goal is to fully automate the warehouse management in a large scale, we believe battery-powered active tags are a better choice. Their much longer operational distance allows a reader to access numerous tags in a large area at one or a few fixed locations.

The deployment of active tags will not only make the objects in a warehouse wirelessly identifiable, but also provide an "infrastructure" that we can leverage to do other things. For example, we may incorporate miniaturized sensors into a tag's circuit to collect useful information in real time and report the information to the RFID reader periodically. Such enhancement to the RFID system will tremendously expand its utility. A sensor may be designed to monitor the state of the tag itself, for instance, the residual energy level of the battery. In this case, the information reported to the reader can be just one bit: '1' indicates that the battery needs to be replaced, and '0' means otherwise. In another example, consider a large chilled food storage facility, where sensor-augmented RFID tags are attached to the food items. Periodic temperature readings help ensure the quality of the food.

The problems of how to build sensor-augmented tags or how to encode sensor information in a number are outside the scope of this paper. We study how to design efficient protocols to collect sensor-produced information from the tags. Information collection is a well-studied area in sensor networks, but RFID systems are fundamentally different. They use a low-rate channel (53Kb/sec in [1]). Communications are driven by the reader. All tags communicate directly with the reader, and they do not communicate amongst themselves. On the contrary, sensor networks in the literature are mostly modeled as multi-hop wireless networks that are capable of executing a rich set of routing, packet scheduling, data aggregation, localization, security, and MAC protocols. These protocols do not exist for RFID tags, and in fact they cannot be handled by the limited resources on the tags. The simpler communication model adopted by RFID systems changes the nature of the information collection problem as it places more restrictions in the solution space.

The industrial RFID research strives to design efficient circuit to operate for a greater distance and better reliability at a lower power level. Much of the academic research is devoted to design *ID-collection protocols* that read the IDs from the tags [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. In recent years, some interest is shifted to other functions that do not require reading the tag IDs, such as estimating the number of tags in the system [12], [13], [14], [8], [15], [16]. An almost-universal performance objective of these protocols is to reduce the overall time it takes to read tag IDs or perform other functions. The reason is that a RFID system in a large warehouse may have tens of thousands of tags and, because it uses a low-rate channel, the execution time for completing a task can be very long. A long execution time is not desirable in a busy warehouse environment. Moreover, continuously powering the tags for an extended period will drain their batteries. Some interesting work also exists for the security of RFID systems [17], [18], [19].

As we briefly reviewed above, a rich set of research results exists for various important problems in RFID systems. Using RFID tags for sensing purpose is not new [20], [21]. But to the best of our knowledge, there is no prior work on how to efficiently collect sensor-produced information from a large number of tags to a RFID reader. Our main objective is to solve this problem in approximately optimal (minimum) time.

In this paper, we first give a lower bound on the execution time for any sensor information collection protocol. We point out that the existing ID-collection protocols are ill-fitted for this task. We then present a straightforward polling-based protocol as a baseline for comparison. Its execution time is much larger than the lower bound and its energy cost is also very high. We set forward to design more sophisticated protocols that significantly reduce the execution time toward the lower bound. The first protocol is called the single-hash information collection protocol (SIC). It totally eliminates radio collisions by using a hash function to assign tags to unique slots of a frame, during which the tags can transmit their data. However, a majority of the slots in the frame has to be wasted due to hash collisions. Our second protocol is called the multi-hash information collection protocol (MIC). It uses multiple hash functions to resolve hash collisions. By using seven hash functions, the protocol's execution time is within 1.61 times the lower bound. For some applications, our simulations show that the execution time of the MIC protocol is about one eighth of the polling-based protocol and one twentieth of an ID-collection protocol that is modified to piggyback sensor information.

## II. System Model and Problem Statement

### A. Problem

Consider a RFID system with a large number of active tags deployed in a region. Each tag is equipped with a sensor that generates a certain type of information, which can be one bit or multiple bits. In the rest of the paper, we will refer to a tag's sensor information simply as a tag's information. We assume that the RFID reader and the tags transmit with sufficient power such that they can communicate over a long distance. Communications between the reader and the tags are time-slotted. The reader's signal will synchronize the clocks of the tags. Generally speaking, communications are driven by the reader in a request-and-response pattern. The reader issues a request, which is followed by a tag's response or a slotted time frame in which multiple tags respond.

The problem is to design a protocol for a reader to periodically collect information from the tags. Our goal is to minimize the execution time of the information collection protocol, so that it uses as little time as possible to gather data from the tags.

### B. Assumption

We assume that the RFID reader has access to a database that stores the IDs of all tags. This is a reasonable assumption for RFID-assisted warehouse management, where the tag IDs are read into a database when new objects are moved into the system and they are removed from the database when the objects are moved out. Even if this operation is not performed, there are many protocols that are designed to collect all tag IDs from the system (see the introduction). Once the tag IDs are collected, we can use the protocols designed in this paper to periodically collect information from the tags.

The set of tags in a warehouse changes over time. Because the execution time of our protocols is small, the set of tags is likely to be stable during the protocol execution. However, even if the set of tags changes, the reader can simply ignore the tags that are added or removed from the system during the protocol execution. The reader will start to collect information from new tags in the next execution of the protocol.

Actions may need to be taken after the sensor information indicates a problem: For example, the battery of a sensor needs to be replaced or the temperature in a certain section of a chilled storage is too high. The problem of physically locating the alarm-raising tag is beyond the scope of this paper. One possible method is to instruct the tag to keep transmitting so that it can be located by a mobile device that detects the direction and distance of a transmitting target. Another approach is to use a localization protocol [22].

### C. Performance Lower Bound and ID-collection Protocols

Let $t_{id}$ be the length of a time slot that the reader uses to broadcast a tag ID, which is 96 bits in the Gen2 standard. Note that the amount of time it takes the reader to transmit an ID may be different from what it takes a tag to transmit an ID because the reader and the tags may operate at different transmission rates [1]. Let $t_{inf}$ be the length of a time slot for a tag to transmit its information. The value of $t_{inf}$ depends on the number of bits that the information contains, which is application-specific. Let $n$ be the number of tags in the system. A lower bound for any protocol to collect information from all tags is $n \times t_{inf}$, which is the aggregate time for all tags to report their information. This lower bound is not achievable because it takes additional time for the reader to send its request(s). However, we can design a protocol whose expected execution time is reasonably close to this lower bound.

Collecting sensor information from tags is a different problem than collecting IDs from the tags. In fact, solutions to these two problems are complementary in practice. First, the ID of a tag only needs to be read once when the tag enters the system and it is removed when the tag exits. Sensor information needs to be collected periodically. Second, tag IDs are a set of numbers, whereas sensor information is not only a set of sensor readings but also a mapping from the readings to the tags where each sensor reading takes place.

One may argue that an ID-collection protocol can piggyback a tag's sensor information when it reads the tag's ID. There are two major types of ID-collection protocols: ALOHA-based or tree-based. It is well known that, for any ALOHA-based protocol [6], [7], [8], [9], [10], [11], the optimal execution time for reading $n$ tags is $e \times n \times T$ [23], where $e$ is the natural constant and $T$ is the length of a time slot in which a tag's ID and its sensor information can be transmitted.[1] Note that $n \times T$ is not achievable due to collision in ALOHA. For tree-based protocols [2], [3], [4], [5], analytical and simulation results have shown that their best performance is comparable to the best of the ALOHA-based protocols.

We know that a lower bound for only collecting sensor information from $n$ tags is $n \times t_{inf}$, where $t_{inf}$ can be as little as one seventh of $T$ when one bit information is reported (see Section VI). Hence, the optimal execution time $e \times n \times T$ of an ID-collection protocol can be almost twenty times of our lower bound. In contrast, our best protocol specifically designed for information collection achieves an execution time within 1.61 times the lower bound. The reason is that when we periodically collect sensor information from tags, the IDs of the tags are supposed to be already known and in fact our protocol design relies on the knowledge of these IDs to help avoid radio collisions in order to improve time efficiency.

## III. Polling-based Information Collection Protocol

Our baseline protocol is called the *polling-based information collection protocol* (PIC). It is very simple. The RFID reader broadcasts the tag IDs one after another. After it transmits an ID, it waits for a period of $t_{inf}$ to receive the information of the corresponding tag. Hence, the time to collect information from one tag is $t_{id} + t_{inf}$. The total execution time of the protocol for collecting information from all tags is $n \times (t_{id} + t_{inf})$.

PIC has two major limitations. First, its execution time is much larger than the optimal value $n \times t_{inf}$. Using the parameters in [1], we find that $t_{id}$ can be twelve times of $t_{inf}$ when the information that a tag reports is one bit. Hence, $n \times (t_{id} + t_{inf})$ is up to thirteen times of the lower bound, which leaves much room for improvement. Second, each tag must continuously listen to the communication channel until its ID is received. If battery-powered active tags are used, this will cause significant energy overhead because each tag has to keep powering its circuit and may have to receive thousands of tag IDs before finding its own. In the next two sections, we propose two protocols that solve the energy problem and are much more time-efficient.

## IV. Single-hash Information Collection Protocol

In this section, we propose a Single-hash Information Collection protocol (SIC) that avoids the transmission of tag IDs and does not require the tags to continuously listen to the channel.

### A. Protocol Overview

The execution of the SIC protocol consists of multiple phases. Every phase has the same structure: It begins with *an information collection request* sent from the reader to the tags, followed by a *slotted time frame*, in which some tags are scheduled to transmit their information. As we will explain later, about 36.8% of the tags are scheduled for transmission in the first phase, 36.8% of the remaining tags are scheduled in the second phase, ..., until all tags are scheduled for transmission. We stress that each tag will be scheduled only once in one of the phases, and it will be *assigned* by the reader to a unique slot in the time frame of that phase.

In the following, we first explain how to assign tags to slots, and then give the protocol details.

### B. Assigning Tags to Time Slots Using a Hash Function

Consider an arbitrary phase. Suppose there are $n'$ tags from which the reader has not yet received information. Only these tags are considered for slot assignment because the information of other tags has been received in the previous phases. Clearly, $n' = n$ for the first phase.

The reader always sets the number of slots in the frame equal to the number of tags it considers for slot assignment. Namely, the frame size is $n'$. Before the reader sends out a request, it has to determine which tags should transmit in this phase and which slots in the frame they should be assigned to. To avoid collision, it should never assign more than one tag to a slot. Because the number of tags is equal to the number of slots, it is not difficult for the reader to construct a one-to-one mapping from the tags to the slots. But it is too costly to inform the tags about this mapping, especially when the set of tags may change each time the protocol is executed.

Our solution is for the reader to use a hash function $H$ to map the tags to the slots, while the tags use the same hash function to determine which slots they should use. The hash function takes the ID of a tag and a random number $r$ as input and produces a pseudo random number $H(ID, r)$ as output, which is used as the slot index that the tag is mapped to. However, this approach does not ensure one-to-one mapping. Multiple tags may be mapped to the same slot. In this case, these tags cannot transmit in the slot because otherwise collision would occur. The slot is thus *wasted*. If no tag is mapped to a slot, that slot is also *wasted*. Only when one and only one tag is mapped to a slot, the reader will assign the tag to the slot. In this case, we say the slot is *useful*. How to inform tags which slots are useful so that the tags assigned to them will transmit in these slots? We introduce an indicator vector, which is described in the next subsection.

### C. Protocol Description

SIC consists of multiple phases. In each phase, the reader sends out a request and then tags transmit their information in the subsequent frame. Before sending out the request, the RFID reader has to assign tags to the slots of the frame. It picks a random number $r$ and uses the hash function to map the IDs of the tags to the slots. After determining which slots are useful and which have to be wasted, the reader constructs an $n'$-bit *indicator vector*, where $n'$ is the number of tags that are considered for slot assignment. Recall that it is also the number of slots in the frame. Each bit in the vector corresponds to a slot in the frame at the same index location. If the slot is useful

[1]In an ALOHA-based protocol, we cannot let tags only transmit their sensor readings without sending their IDs because we need to know which tag each senor reading belongs to.

(i.e., one and only one tag is mapped to it), the bit value is 1; otherwise, it is 0.

The request broadcast by the reader consists of the information type to be reported, the frame size (i.e., number of slots in the frame), a random number $r$, and the indicator vector, where $r$ is used by the hash function and it is different in each phase. If the vector is too long, the reader divides it into segments of 96 bits (equivalent to the length of a tag ID) and transmits each segment in a time slot of length $t_{id}$.

Using the same hash function, a tag knows the index $i$ of the slot it is mapped to. After the tag receives the request, it knows whether its slot will be useful or not by examining the $i$th bit in the indicator vector. If the bit is 0, the tag will not transmit. If the bit is 1, the tag will transmit its information during the $i$th slot in the frame and it will not participate in the remaining phases.

It should be noted that the tag does not have to receive the whole indicator vector. It knows the index $i$ of the bit it looks for. Hence, it also knows which segment of the indicator vector it must receive. The tag can be in a stand-by mode to conserve energy at times other than when it receives its segment of the indicator vector or transmits its information.

The first phase considers $n$ tags for slot assignment and its frame has $n$ slots. Each subsequent phase considers a fewer number of tags and has a smaller frame accordingly. The protocol terminates after all tags report their information. Alternatively, the reader may stop the SIC protocol when the number of remaining tags is fewer than a small threshold, and then it invokes the PIC protocol to collect information from these tags.

The implementation of the hash function will be discussed in Section V-E, and the impact of channel error will be considered in Section V-G.

### D. Expected Execution Time

We derive the expected execution time of the SIC protocol. Consider an arbitrary tag $x$ and an arbitrary phase that $x$ participates. Let $n'$ be the number of tags that are considered for slot assignment in this phase. The frame size is also $n'$. Let $P_1$ be the probability that no other tag is mapped to the slot that $x$ is mapped to. The subscript '1' indicates that a single hash function is used. The purpose will be clear when we discuss multiple hash functions in the next section.

$$P_1 = (1 - \frac{1}{n'})^{n'-1} \approx e^{-\frac{n'-1}{n'}} \approx e^{-1} \approx 36.8\%, \qquad (1)$$

where $e$ is the natural constant. When this happens, tag $x$ will be assigned to the slot. It will not participate in the remaining phases. Hence, the expected number of phases that tag $x$ participates is

$$\begin{aligned} & 1 \times P_1 + 2 \times (1 - P_1)P_1 + 3 \times (1 - P_1)^2 P_1 + ... \\ & = \sum_{i=0}^{\infty} P_1(1 - P_1)^i + \sum_{i=1}^{\infty} P_1(1 - P_1)^i + \sum_{i=2}^{\infty} P_1(1 - P_1)^i + ... \\ & = 1 + (1 - P_1) + (1 - P_1)^2 + ... \\ & = \frac{1}{P_1} \approx e, \end{aligned}$$

where we have used the fact that $\sum_{i=0}^{\infty} P_1(1-P_1)^i = 1$. Recall that in any phase the number of slots in the time frame is equal to the number of tags considered for slot assignment. In other words, each time $x$ participates in a phase, its presence contributes a slot in the frame. Overall, the expected number of slots in all phases that can be attributed to $x$'s participation is $e$. The total number of tags in the system is $n$. Therefore, the number of slots in the frames of all phases is expected to be $n \times e$. The expected frame time in all phases is $e \times n \times t_{inf}$.

There is a one-to-one correspondence between bits in an indicator vector and slots in a frame. Hence, the total number of bits in all indicator vectors is also $n \times e$. The expected time for transmitting all indicator vectors is $\frac{e \times n}{96} \times t_{id}$. Due to the large denominator of 96, it is smaller than the total frame time, $e \times n \times t_{inf}$. The rest of the information collection request excluding the indicator vector is very small and can be ignored. Hence, the expected execution time of SIC is $e \times n \times t_{inf} + \frac{e \times n}{96} \times t_{id}$. The first item is about 2.72 times of the lower bound $n \times t_{inf}$.

We go back to (1). Out of the $n'$ slots in a frame, the number of useful slots is $n'P_1 \approx 36.8\%n'$. Hence, in each phase of the SIC protocol, only about 36.8% of the time slots are useful and 63.2% of the slots are wasted. This gives us significant room for further improvement, which leads to our final protocol in the next section.

## V. Multi-hash Information Collection Protocol

We propose a Multi-hash Information Collection protocol (MIC) to solve the hash collision problem of SIC.

### A. Protocol Overview

MIC is similar to SIC except that it assigns tags to slots using $k$ hash functions in order to alleviate the problem of wasted slots. More specifically, it hashes each tag to $k$ slots in the frame. As long as any one of these slots has no other tag, the reader is able to assign the tag to the slot.

The execution of the MIC protocol also consists of multiple phases. In each phase, the RFID reader broadcasts an information collection request that is followed by a slotted time frame, in which some tags transmit their information.

In the following, we first explain how to assign tags to slots by using $k$ hash functions. We then introduce a mechanism (called hash-selection vector) to inform the tags about the assignment.

### B. Assigning Tags to Slots Using Multiple Hash Functions

The $k$ hash functions are denoted as $H[i]$, $1 \leq i \leq k$, which takes a tag ID and a random number $r$ as input and produces a pseudo-random number $H[i](ID, r)$ as output.

Consider an arbitrary phase in the execution of MIC. Let $n'$ be the number of tags for slot assignment in this phase. The frame size is also set to be $n'$. If it is the first phase, $n' = n$.

The slot assignment consists of $k$ rounds, each involving one hash function. In the first round, we apply $H[1]$ to map the tags to the slots. A tag is assigned to a slot if it is the only one that is mapped to the slot. After assignment, we remove the tag from being further considered in the remaining rounds that involve other hash functions. We also mark the slot as being *occupied*. The slots that are not marked at the end of this round are said to be *unoccupied*.

In the second round, we apply $H[2]$ to map the remaining tags to the $n'$ slots. If a tag is mapped to an unoccupied slot and it is the only one that is mapped to the slot, it is assigned to the slot and removed from further consideration. The slot is marked as occupied.

The above process is repeated for other hash functions one round after another in order to assign as many tags as possible to the unoccupied slots. An illustrative example is given in Fig. 1 (a)-(b). After all $k$ hash functions are used, the reader has a subset of tags that are assigned to slots, and it also know which hash function each of these tags should use. The problem of communicating this information to the tags will be addressed shortly.

Please be aware of the difference between the term "map" and the term "assign". A tag may be mapped to multiple slots based on the $k$ hash functions, but it can only be assigned to one slot. Also be aware of the difference between "phase" and "round". Each execution of MIC consists of phases. Each phase is a request-and-response exchange between the reader and the tags. Before that exchange, the reader has to assign tags to slots and the assignment process consists of multiple rounds when more than one hash function is used.

## C. Protocol Description

In the first phase of the MIC execution, before sending out an information collection request, the RFID reader determines which tags are assigned to which slots (see the previous subsection). It then constructs an $n$-element *hash-selection vector*. Each element in the vector corresponds to a slot in the frame at the same index location. If no tag is assigned to a slot, the reader sets the corresponding element in the hash-selection vector to zero. If a tag is assigned to a slot using the $j$th hash function, the reader sets the corresponding element to be $j$. The size of an element is $\lceil \log_2(k+1) \rceil$ bits.

The request broadcast by the reader consists of the information type to be reported, the frame size, a random number $r$, and the hash-selection vector, where $r$ is used by the hash functions and it is different in each phase. The hash-selection vector is divided into segments of 96 bits (equivalent to the length of a tag ID), and each segment is transmitted in a time slot of size $t_{id}$.

The tags will receive the hash-selection vector along with other information in the request. Using the same $k$ hash functions, each tag knows which $k$ slots in the frame and which $k$ elements in the hash-selection vector it is mapped to. If a tag is assigned to a slot, it must be one of those $k$ slots. If a tag is assigned to a slot by the reader using the $j$th hash function, the corresponding element in the hash-selection vector must have a value of $j$ because this is exactly how the hash-selection vector is constructed. The inverse is also true. *If a tag finds that (1) it is mapped to a slot $s$ using the $j$th function and (2) the corresponding element in the hash-selection vector is also $j$, then it can conclude that it must have been assigned to slot $s$ by the reader. If multiple hash functions satisfy the above conditions, the tag only uses the one that has the smallest value of $j$.* See Section V-F for correctness proof. An illustrative example is given in Fig. 1.

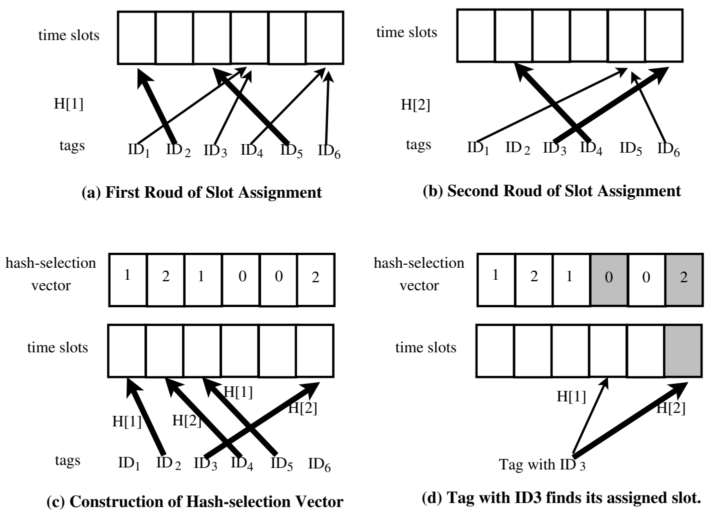

Fig. 1. Arrows represent the mapping from tags to slots based on hash functions. Among them, thick arrows represent the assignment of tags to slots. In this example, $k = 2$. (a) Two tags, $\text{ID}_1$ and $\text{ID}_5$, are assigned to slots in the first round when $H[1]$ is applied. (b) Two more tags, $\text{ID}_3$ and $\text{ID}_4$, are assigned in the second round when $H[2]$ is applied. (c) The reader constructs a hash-selection vector based on the slot assignment. (d) After receiving the vector, the tag with $\text{ID}_3$ examines the elements in the hash-selection vector that it is mapped to. The element mapped by $H[2]$ has a value of 2. The tag knows that it must be assigned to the corresponding slot.

Hence, in order to determine whether it is assigned to a slot, a tag only needs to examine the elements in the hash-selection vector that it is mapped to by the $k$ hash functions. Let $E_j$, $1 \leq j \leq k$, be the element that the tag is mapped to by the $j$th hash function. The tag examines the elements in order from $E_1$ to $E_k$. If it finds the value of an element $E_j$ is equal to $j$, the tag knows that it must be assigned to a slot by the reader using the $j$th hash function. In this case, it will stop examining the remaining elements and wait until the assigned slot arrives. It will transmit during that slot and then stop participating further in the protocol execution.

Note that a tag does not have to receive the whole hash-selection vector. It knows the indices of the elements it looks for. The tag can be in a stand-by mode to conserve energy at times other than it receives its segments of the hash-selection vector or transmits its information.

After the first phase completes, the RFID reader moves to the second phase, which is identical except that the reader removes the tags for which it has assigned slots. It only considers the tags that have not got a chance to transmit. The frame size in this phase is reduced accordingly.

The above process repeats phase after phase until all tags report their information. Alternatively, the reader may stop when the number of remaining tags is fewer than a small threshold, and it invokes the PIC protocol to collect information from these tags.

Clearly, SIC is a special case of MIC when $k = 1$.

## D. Expected Execution Time

To compute the expected execution time of MIC, we need to find the number of phases that an arbitrary tag is expected to participate. Consider an arbitrary tag $x$ and an arbitrary phase that $x$ participates. Let $n'$ be the number of tags that participate in this phase. The frame size is also $n'$. Let $P_i$ be the probability that tag $x$ is assigned to a slot after the first $i$ hash functions are applied. That is, $P_i$ is the probability that one of the first $i$ hash functions maps $x$ to a slot that no other tag is mapped to. When this happens, tag $x$ will transmit in this phase and will stop participating in the remaining phases.

From (1), we know that $P_1 = (1 - \frac{1}{n})^{n-1} \approx e^{-1}$. Next, we derive a recursive formula for $P_i$, $i > 1$. After the first $i-1$ hash functions are applied, there are two cases. The *first case* is that tag $x$ has been assigned to a slot by one of those $i-1$ hash functions. The probability for this to happen is $P_{i-1}$. The *second case* is that tag $x$ has not been assigned to any slot and thus it will be considered when the $i$th hash function is applied. The probability for this to happen is $1 - P_{i-1}$. We focus on the second case below.

Because the number of tags is the same as the number of slots and the tag-to-slot assignment is one-to-one mapping, the probability for an arbitrary slot to stay unoccupied after $i-1$ hash functions is the same as the probability for an arbitrary tag to stay unassigned, $1 - P_{i-1}$. When the $i$th hash function is applied, the slot that tag $x$ is mapped to has a probability of $1 - P_{i-1}$ to be unoccupied. For each of the other $n'-1$ tags, it has a probability of $1 - P_{i-1}$ to be unassigned and, if so, it has a probability of $\frac{1}{n'}$ to be mapped to the same slot as $x$ does. Hence, the probability $p$ for tag $x$ to be the only one that is mapped to an unoccupied slot is

$$\begin{aligned} p &= (1 - P_{i-1})(1 - (1 - P_{i-1})\frac{1}{n'})^{n'-1} \\ &\approx (1 - P_{i-1})e^{-(1-P_{i-1})} \end{aligned} \tag{2}$$

Recall that we are considering the second case here. Combining both cases discussed previously, we have

$$\begin{aligned} P_i &= P_{i-1} + (1 - P_{i-1}) \times p \\ &= P_{i-1} + (1 - P_{i-1})^2 e^{-(1-P_{i-1})}, \end{aligned} \tag{3}$$

where the first item on the right side is the probability for a tag to be assigned to a slot by one of the first $i-1$ hash functions and the second item is the probability for the tag to be assigned to a slot by the $i$th hash function. The probability for tag $x$ to be assigned to a slot after all $k$ functions are applied is $P_k$, and this is the case for any phase that $x$ participates.

Based on the recursive formula in (3), we compute the numerical values of $P_i$ in Table I, which match perfectly with our simulation results in Section VI. If seven hash functions are used, i.e., $k = 7$, the probability for an unassigned tag to be assigned to a slot in any phase is $P_7 \approx 86.1\%$. The probability for an arbitrary slot to be useful is also 86.1%. Only 13.9% of the slots in each frame is wasted.

The expected number of phases that tag $x$ participates is

$$1 \times P_k + 2 \times (1 - P_k)P_k + 3 \times (1 - P_k)^2 P_k + ... = \frac{1}{P_k}.$$

TABLE I
NUMERICAL VALUES OF $P_i$

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ |
|---|---|---|---|---|---|---|
| 36.8% | 58.0% | 69.6% | 76.4% | 80.8% | 83.9% | 86.1% |

For each phase that $x$ participates, the reader allocates a slot in the frame. Hence, the expected number of slots that can be attributed to $x$'s participation in the protocol is $\frac{1}{P_k}$. There are $n$ tags. The total number of slots in all phases is expected to be $\frac{n}{P_k}$. The total expected time in the frames of all phases is $\frac{n}{P_k}t_{inf}$. When $k = 7$, it becomes $1.16 \times n \times t_{inf}$, just 16% more than the lower bound. Because 32 elements of the hash-selection vector can fit in a segment of 96 bits, the expected time for all indicator vectors is $\frac{n}{32P_k}t_{id}$. Hence, the expected execution time of MIC is about $\frac{n}{P_k}t_{inf} + \frac{n}{32P_k}t_{id}$.

The ratio of $\frac{n}{32P_k}t_{id}$ to the lower bound $n \times t_{inf}$ is largest when the information reported by each tag is one bit. In this case, $t_{id}$ is about 12 times of $t_{inf}$, according to the parameters in [1] (see details in Section VI). Hence, when $k = 7$, $\frac{n}{32P_k}t_{id}$ is up to 45% of the lower bound. Consequently, the expected execution time of MIC is up to 1.61 times the lower bound.

## E. Hash Functions

There are many efficient hash functions in the literature. We describe a simple implementation that helps to keep the complexity of a tag's circuit low. The tags do not have to fully implement the $k$ hash functions, $H[i](ID, r)$. When $k = 1$, the expected number of phases a tag will participate is just $\frac{1}{P_1} \approx 2.7$, which means that a tag only needs to produce 2.7 hash values on average. Similarly, when $k = 3$, a tag needs $\frac{1}{P_3} \approx 1.4$ hash values on average. When $k = 7$, a tag needs $\frac{1}{P_7} \approx 1.2$ hash values on average. For $n = 50,000$, each hash value is 16 bits long. We may derive these hash values from a ring of pre-stored random bits as follows: We use an offline random number generator with the ID of a tag as seed to generate a string of random bits. We take $k$ segments of the string. Each segment contains a certain number of bits, forming a ring by logically connecting the last bit with the first bit. These rings are pre-stored in the tags before they are deployed. To find the value of $H[i](ID, r)$, a tag takes a certain number of bits from the $i$th ring. More specifically, it takes a number of bits from the ring clockwise beginning from the $r$th bit. An alternative approach is to begin from the $r$th bit and take one bit after every $r$ bits until a sufficient number of bits are taken. The final hash value is the number represented by these bits modulo the frame size.

The larger the size of each ring, the better the pseudo randomness in the hash output. Because our protocol only requires a tag to produce a very small number of hash values from each ring, a ring size of 100 bits should be more than sufficient. In this case, when $k = 3$, each tag needs to store 300 bits to implement the hash functions. When $k = 7$, each tag needs 700 bits.

The RFID reader knows the IDs of the tags, and it picks the random number $r$. Hence, it can predict the hash values of all tags.

## F. Correctness

In Section V-B, the RFID reader assigns a tag to a slot only when no other tag is mapped to the same slot. After a tag is assigned to a slot, the reader removes it from further consideration. Hence, from the reader's point of view, each tag is uniquely assigned to a slot. According to the protocol description in Section V-C, each tag will only transmit once. What we want to make sure is that the tag will transmit in the assigned slot. Moreover, there should not be collision in that slot.

To determine in which slot it transmits, a tag first uses the $k$ hash functions to map itself to $k$ slots in the frame. The rule states that: *If the tag finds that (1) it is mapped to a slot $s$ using the $j$th function and (2) the corresponding element in the hash-selection vector is also $j$, then it can conclude that it must have been assigned to slot $s$ by the reader. If multiple hash functions satisfy the above conditions, the tag only uses the one that has the smallest value of $j$.* When a tag $x$ is mapped to a slot using the $j$th hash function, if the tag finds that the corresponding element in the hash-selection vector is also $j$, it means that the reader has assigned a tag to the slot based on the $j$th function, and moreover the reader will do so only when a single tag is mapped to the slot using the $j$th function. Tag $x$ can thus conclude that this single tag must be itself. Because it is the only tag that will transmit in this slot, there will not be a collision.

## G. Channel Error

We now consider the impact of channel error. If a segment in the hash-selection vector is corrupted, the tags that extract information from that segment may transmit in wrong slots, causing collision. It may also happen that the information transmitted by a tag in the correct slot is corrupted by noise in the channel. If the channel error is mild and the application can tolerate a certain level of error, the protocol may not need additional error control mechanisms. For example, suppose the reader collects the battery status of the tags to see if any battery needs to be replaced. The reader may periodically collect such information. Over a period of time, it will receive a certain number of readings from each tag. It decides whether a tag needs to replace battery based on the majority votes. In this way, occasional information corruption due to channel error does not cause a misjudgement of the battery status. In another example, consider a large chilled food storage facility and the application is to monitor the temperature at each section of the storage by using sensor-augmented RFID tags that are attached to the food items. Each section has many tags, which provide a large amount of redundancy in the information reported to the RFID reader. If temperature readings from some tags are corrupted, the reader can still retrieve correct temperature data by removing outliers from all the readings it receives from a particular section of the storage.

If the application requires that the information received from every tag is correct, we need to add checksum such as a CRC code to each transmission for error detection. Each segment of 96 bits in the hash-selection vector carries 16-bit checksum, and it uses the remaining 80 bits to carry 26 elements of 3 bits each. Each information report from a tag also carries 16-bit checksum. Consider the following two cases: (1) When a tag finds that one of its $k$ segments in the hash-selection vector is corrupted, it ignores the segment and only uses other segments to decide whether it is assigned to a slot. If none of the other segments suggests that it is assigned to a slot, it makes the conservative decision that it will not participate in the remaining phases even through it does not transmit in this phase, because the reader might have assigned it to a slot using the corrupted segment. (2) Beside the case that a tag may not transmit at all, even when a tag transmits in the correct slot, that slot may be corrupted due to channel error, which can be detected by the reader through the mismatching CRC code.

The reader handles the above cases in the same way: It only assigns one slot to each tag during the execution of MIC. If it does not receive a tag's information in the assigned slot or the information is corrupted, it will not assign another slot because otherwise we would run into the issue of acknowledging the tags whether their information is received correctly — this can get complicated, considering that the acknowledgement itself may also be corrupted. After MIC completes, the reader wakes up all tags and performs the polling protocol (PIC) on the set of tags from which the information has not been received correctly. We expect the set to be relatively small in a practical environment where the channel noise is not too large to hinder the effectiveness of the RFID system. In PIC, each transmission also carries CRC. Due to channel error, the reader may poll for the information of a tag more than once until the information is correctly received.

The overhead for the above error control will be investigated by simulations in the next section.

## VI. Simulation Results

### A. Simulation Setting

The simulation setting is based on the Philips I-Code specification [1]. Any two consecutive transmissions (from the reader to tags or vice versa ) are separated by a waiting time of 302 $\mu s$. According to the specification, the transmission rate from a tag to the reader is different than the transmission rate from the reader to a tag. The rate from a tag to the reader is 53Kb/sec; it takes 18.88 $\mu s$ for a tag to transmit one bit. The value of $t_{inf}$ is calculated as the sum of a waiting time and the time for transmitting the information, which is 18.88 $\mu s$ multiplied by the length of the information. For example, if the sensor information is one bit, $t_{inf}$ is 321 $\mu s$ if a CRC code is not added, and it is 623 $\mu s$ if a 16-bit CRC is added. If the sensor information is 16 bits, $t_{inf}$ is 604 $\mu s$ without CRC, and it is 906 $\mu s$ with CRC.

The transmission rate from the reader to tags is 26.5 Kb/sec; it takes 37.76 $\mu s$ for the reader to transmit one bit. Each tag ID contains 96 bits, which include a 16-bit CRC code according to the Gen2 standard. Recall that $t_{id}$ is the time it takes the reader to transmit an ID to a tag. It is 3927 $\mu s$ (including a waiting time before the transmission). The time for the reader to transmit a segment of the indicator vector or a segment of the

TABLE II
EXECUTION TIME COMPARISON (IN SECONDS) WHEN THE SENSOR INFORMATION IS 1 BIT LONG.

| | n = 10,000 | 30,000 | 50,000 | 70,000 | 90,000 |
|---|---|---|---|---|---|
| **EDFSA** | 89.4 | 268.8 | 530.1 | 749.0 | 1051.1 |
| **PIC** | 42.5 | 127.4 | 212.4 | 297.3 | 382.3 |
| **SIC** | 10.0 | 30.0 | 49.7 | 69.6 | 89.2 |
| **MIC, k = 3** | 5.9 | 17.4 | 29.0 | 40.6 | 52.1 |
| **MIC, k = 7** | 5.2 | 15.5 | 25.8 | 36.1 | 46.4 |
| **lower bound** | 3.2 | 9.6 | 16.0 | 22.5 | 28.9 |

TABLE III
EXECUTION TIME COMPARISON (IN SECONDS) WHEN THE SENSOR INFORMATION IS 16 BITS LONG.

| | n = 10,000 | 30,000 | 50,000 | 70,000 | 90,000 |
|---|---|---|---|---|---|
| **EDFSA** | 96.9 | 292.3 | 576.7 | 814.8 | 1156.5 |
| **PIC** | 45.3 | 135.9 | 226.6 | 317.2 | 407.8 |
| **SIC** | 17.7 | 53.3 | 88.6 | 124.0 | 159.0 |
| **MIC, k = 3** | 9.9 | 29.6 | 49.3 | 69.0 | 88.7 |
| **MIC, k = 7** | 8.5 | 25.4 | 42.3 | 59.2 | 76.0 |
| **lower bound** | 6.0 | 18.1 | 30.2 | 42.3 | 54.4 |

TABLE IV
EXECUTION TIME COMPARISON (IN SECONDS) WHEN THE SENSOR INFORMATION IS 32 BITS LONG.

| | n = 10,000 | 30,000 | 50,000 | 70,000 | 90,000 |
|---|---|---|---|---|---|
| **EDFSA** | 106.4 | 320.5 | 623.6 | 884.8 | 1261.7 |
| **PIC** | 48.3 | 145.0 | 241.7 | 338.3 | 435.0 |
| **SIC** | 25.9 | 78.3 | 130.1 | 182.0 | 233.4 |
| **MIC, k = 3** | 14.3 | 42.6 | 71.0 | 99.4 | 127.8 |
| **MIC, k = 7** | 12.0 | 35.9 | 59.8 | 83.7 | 107.6 |
| **lower bound** | 9.1 | 27.2 | 45.3 | 63.4 | 81.6 |

hash-selection vector is the same. However, if a tag transmits a 96-bit ID to the reader, it only takes 2114 $\mu s$ due to a different transmission rate.

In each simulation run, we set the number $n$ of tags in the RFID system. We then execute five protocols: PIC, SIC, MIC with $k = 3$, MIC with $k = 7$, and EDFSA [6]. EDFSA is one of the best ID-collection protocols. We modify it for information collection. The modification is simple: When a tag transmits its ID to the reader, it piggybacks its sensor information. We measure and compare the execution times of the five protocols. Each data point in the figures is the average outcome of 100 simulation runs under the same setting.

In the following, we first present the simulation results when CRC codes are not used for error control, and then we present the results when CRC codes are used (see Section V-G).

## B. Execution Time Comparison

We first study the performance of our protocols without considering channel error. That is, the RFID reader can always correctly receive the information when a tag transmits in a slot without collision.

Table II compares the execution times of the five protocols under different values of $n$ when the sensor information is one bit. This corresponds to the application of monitoring the battery status of the RFID tags: '1' means the battery is ok; '0' means the battery needs to be replaced. We examine the fourth column in the table for $n = 50,000$ (imagine that a large military base stores 50,000 pieces of weapons and ammunition packets, each attached with a tag). The execution time of EDFSA is 530.1 seconds, which is about thirty-three times of the lower bound, 16.0 seconds. PIC reduces the execution time by 60% to 212.4 seconds because it eliminates collisions that exist in EDFSA, which is ALOHA-based. SIC further reduces the time to 49.7 seconds, about one fourth of the time needed by PIC. Our best protocol, MIC, reduces the execution time to 29.0 seconds when three hash functions are used or 25.8 when seven hash functions are used. Similar conclusions can be drawn from other columns: MIC works the best, SIC follows, then PIC, and finally EDFSA.

Table III and Table IV present the execution times when the sensor information is 16 bits long and 32 bits long, respectively. Again, similar conclusions can be drawn. For example, when the information is 16 bits long and $n = 50,000$, the execution time of MIC with $k = 7$ is 48% of the time needed by SIC, 19% of the time needed by PIC, and just 7.3% of the time needed by EDFSA. When the information is 32 bits long and $n = 50,000$, the execution time of MIC with $k = 7$ is 46% of the time needed by SIC, 25% of the time needed by PIC, and 9.5% of the time needed by EDFSA.

The execution time of MIC with $k = 3$ is only slightly worse than that of MIC with $k = 7$. Because each tag has to store $k$ hash outputs, if one wants to reduce the storage overhead, $k$ may be chosen as 3.

## C. Execution Time Comparison under Channel Error

The method for handling channel error is described in Section V-G. Suppose the sensor information is 1 bit long. Table V, VI and VII present the execution time comparison when the channel error rate is 1%, 5% and 10%, respectively. The channel error rate $c$ is defined as the percentage of slots that is corrupted. In our simulations, each slot has a probability of $c$ to be corrupted. With the presence of different levels of channel error, we continue to observe that MIC performs much better than SIC, which in turn performs better than PIC, which is better than EDFSA.

For example, in Table V where the channel error rate is 1%, when $n = 50,000$, the execution time of MIC with $k = 7$ is 49.4 seconds, the time of SIC is 96.9 seconds, the time of PIC is 232.1 seconds, and the time of EDFSA is 460.0 seconds. In Table VI where the channel error rate is 5%, when $n = 50,000$, the execution time of MIC with $k = 7$ is 69.4 seconds, the time of SIC is 116.8 seconds, the time of PIC is 252.1 seconds, and the time of EDFSA is 480.1 seconds. In Table VII where the channel error rate is 10%, when $n = 50,000$, the execution time of MIC with $k = 7$ is 98.1 seconds, the time of SIC is 145.6 seconds, the time of PIC is 280.8 seconds, and the time of EDFSA is 505.8 seconds.

## D. Values of $P_i$

To verify our analytical results in Section V-D, we measure the values of $P_i$ by simulations. As shown in Table VIII, the

TABLE V
EXECUTION TIME COMPARISON (IN SECONDS) WHEN THE CHANNEL ERROR RATE IS 1%.

| | n = 10,000 | 30,000 | 50,000 | 70,000 | 90,000 |
|---|---|---|---|---|---|
| **EDFSA** | 79.7 | 236.9 | 460.0 | 652.7 | 911.0 |
| **PIC** | 46.4 | 139.3 | 232.1 | 325.0 | 417.8 |
| **SIC** | 19.3 | 58.4 | 96.9 | 135.5 | 173.8 |
| **MIC, k = 3** | 11.3 | 33.9 | 56.5 | 79.0 | 101.6 |
| **MIC, k = 7** | 9.9 | 29.7 | 49.4 | 69.1 | 88.9 |

TABLE VI
EXECUTION TIME COMPARISON (IN SECONDS) WHEN THE CHANNEL ERROR RATE IS 5%.

| | n = 10,000 | 30,000 | 50,000 | 70,000 | 90,000 |
|---|---|---|---|---|---|
| **EDFSA** | 82.0 | 248.5 | 480.1 | 685.0 | 950.4 |
| **PIC** | 50.4 | 151.2 | 252.1 | 352.9 | 453.7 |
| **SIC** | 23.3 | 70.3 | 116.8 | 163.5 | 209.8 |
| **MIC, k = 3** | 15.3 | 45.9 | 76.4 | 106.9 | 137.5 |
| **MIC, k = 7** | 13.9 | 41.6 | 69.4 | 97.1 | 124.8 |

TABLE VII
EXECUTION TIME COMPARISON (IN SECONDS) WHEN THE CHANNEL ERROR RATE IS 10%.

| | n = 10,000 | 30,000 | 50,000 | 70,000 | 90,000 |
|---|---|---|---|---|---|
| **EDFSA** | 86.0 | 259.5 | 505.8 | 715.8 | 995.4 |
| **PIC** | 56.1 | 168.5 | 280.8 | 393.2 | 505.5 |
| **SIC** | 29.0 | 87.5 | 145.6 | 203.7 | 261.6 |
| **MIC, k = 3** | 21.0 | 63.1 | 105.2 | 147.2 | 189.3 |
| **MIC, k = 7** | 19.6 | 58.9 | 98.1 | 137.3 | 176.6 |

TABLE VIII
VALUES OF $P_i$

| | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ |
|---|---|---|---|---|---|---|---|
| **by simulation** | 37.0% | 58.2% | 69.9% | 76.6% | 80.8% | 83.8% | 86.2% |
| **by analysis** | 36.8% | 58.0% | 69.6% | 76.4% | 80.8% | 83.9% | 86.1% |

values of $P_i$ measured from simulations match well with the numerically-computed values based on the recursive formula (3), which confirms the correctness of our analysis.

## VII. CONCLUSION

This paper investigates a new problem of how to efficiently collect sensor information from all tags to a reader in a large RFID system. We present three protocols. The first one, called the polling-based information collection protocol (PIC), serves as a baseline for comparison. The second protocol, called the single-hash information collection protocol (SIC), improves time and energy efficiencies by totally eliminating the transmission of tag IDs. It uses a hash function to assign tags to the slots of a frame, during which the tags can transmit their data successfully. However, due to hash collisions, many slots have to be wasted. A wide gap still exists between the execution time of SIC and a lower bound that we establish. We use multiple hash functions to solve the hash collision problem, which leads to our third protocol, called the multi-hash information collection protocol (MIC). Its execution time is about half of the execution time of the SIC, up to seven times smaller than the execution time of PIC, and up to nineteen times smaller than the execution time of a representative ID-collection protocol [6] that is enhanced to collect sensor information.

## REFERENCES

[1] P. Semiconductors, "I-CODE Smart Label RFID Tags," *http://www.nxp.com/acrobat_download/other/identification/SL092030.pdf*, Jan 2004.
[2] J. Myung and W. Lee, "Adaptive Splitting Protocols for RFID Tag Collision Arbitration," *Proc. of ACM MOBIHOC*, 2006.
[3] N. Bhandari, A. Sahoo, and S. Iyer, "Intelligent Query Tree (IQT) Protocol to Improve RFID Tag Read Efficiency," *Proc. of IEEE ICIT*, 2006.
[4] F. Zhou, C. Chen, D. Jin, C. Huang, and H. Min, "Evaluating and Optimizing Power Consumption of Anti-collision Protocols for Applications in RFID Systems," *Proc. of ISLPED*, 2004.
[5] "Information technology automatic identification and data capture techniques C radio frequency identification for item management air interface - part 6: parameters for air interface communications at 860-960 MHz," *Final Draft International Standard ISO 18000-6*, November 2003.
[6] S. Lee, S. Joo, and C. Lee, "An Enhanced Dynamic Framed Slotted ALOHA Algorithm for RFID Tag Identification," *Proc. of IEEE MOBIQUITOUS*, 2005.
[7] J. R. Cha and J. H. Kim, "Dynamic Framed Slotted ALOHA Algorithms Using Fast Tag Estimation Method for RFID Systems," *IEEE Consumer Communications and Networking Conference (CCNC)*, January 2006.
[8] H. Vogt, "Efficient Object Identification with Passive RFID Tags," *Proc. of IEEE PerCom*, 2002.
[9] V. Sarangan, M. R. Devarapalli, and S. Radhakrishnan, "A Framework for Fast RFID Tag Reading in Static and Mobile Environments," *The International Journal of Computer and Telecommunications Networking*, vol. 52, no. 5, 2008.
[10] B. Zhen, M. Kobayashi, and M. Shimizu, "Framed ALOHA for Multiple RFID Objects identification," *IEICE Transactions on Communications*, Mar 2005.
[11] B. Sheng, Q. Li, and W. Mao, "Efficient Continuous Scanning in RFID Systems," *Proc. of IEEE INFOCOM*, March 2010.
[12] M. Kodialam and T. Nandagopal, "Fast and Reliable Estimation Schemes in RFID Systems," *Proc. of ACM MOBICOM, Los Angeles*, 2006.
[13] M. Kodialam, T. Nandagopal, and W. Lau, "Anonymous Tracking using RFID tags," *Proc. of IEEE INFOCOM*, 2007.
[14] J. Zhai and G. N. Wang, "An Anti-Collision Algorithm Using Two-functioned Estimation for RFID Tags," *Proc. of ICCSA*, 2005.
[15] C. Qian, H. Ngan, and Y. Liu, "Cardinality Estimation for Large-scale RFID Systems," *Proc. of IEEE PerCom*, 2008.
[16] H. Han, B. Sheng, C. C. Tan, Q. Li, W. Mao, and S. Lu, "Counting RIFD Tags Efficiently and Anonymously," *Proc. IEEE INFOCOM*, March 2010.
[17] C. Tan, B. Sheng, and Q. Li, "How to Monitor for Missing RFID Tags," *Proc. of IEEE ICDCS*, June 2008.
[18] T. Li, S. Chen, and Y. Ling, "Identifying the Missing Tags in a Large RFID System," *Proc. of ACM Mobihoc*, 2010.
[19] L. Lu, Y. Liu, and X. Li, "Refresh: Weak Privacy Model for RFID Systems," *Proc. of IEEE INFOCOM*, 2010.
[20] M. Miura, S. Ito, R. Takatsuka, T. Sugihara, and S. Kunifuji, "An Empirical Study of an RFID Mat Sensor System in a Group Home," *Journal of Networks*, vol. 4, no. 2, April 2009.
[21] A. Ruhanen, M. Hanhikorpi, F. Bertuccelli, A. Colonna, W. Malik, D. Ranasinghe, T. S. Lopez, N. Yan, and M. Tavilampi, "Sensor-enabled RFID Tag Handbook," *BRIDGE, IST-2005-033546*, January 2008.
[22] L. M. Ni, Y. Liu, Y. C. Lau, and A. Patil, "LANDMARC: Indoor Location Sensing Using Active RFID," *ACM Wireless Networks (WINET)*, vol. 10, no. 6, November 2004.
[23] L. G. Roberts, "ALOHA Packet System with and without Slots and Capture," *ACM SIGCOMM Computer Communication Review*, vol. 5, no. 2, pp. 28–42, April 1975.