Efficient Object Identification with Passive RFID Tags

Harald Vogt

Department of Computer Science
Swiss Federal Institute of Technology (ETH)
8092 Zürich, Switzerland
vogt@inf.ethz.ch

Abstract. Radio frequency identification systems with passive tags are powerful tools for object identification. However, if multiple tags are to be identified simultaneously, messages from the tags can collide and cancel each other out. Therefore, multiple read cycles have to be performed in order to achieve a high recognition rate. For a typical stochastic anti-collision scheme, we show how to determine the optimal number of read cycles to perform under a given assurance level determining the acceptable rate of missed tags. This yields an efficient procedure for object identification. We also present results on the performance of an implementation.

1 Introduction

Identification is a central concept in user-oriented and ubiquitous computing. Human users are usually identified (and authenticated) by passwords or biometric data. Most applications require some kind of identification in order to deliver personalized information or restrict access to sensitive data and procedures. Object identification, on the other hand, is most useful for applications such as asset tracking (e.g. libraries, animals), automated inventory and stock-keeping, toll collecting, and similar tasks where physical objects are involved and the gap between the physical and the "virtual" world must be bridged. In a world of ubiquitous computing, unobtrusive object identification enables the seamless connection between real-world artifacts and their virtual representations.

Reliable identification of multiple objects is especially challenging if many objects are present at the same time. Several technologies are available, but they all have limitations. Bar codes are the most pervasive technology used today, but reading them requires a line of sight between the reader device and the tag, and often doesn't work without human intervention. Other visual recognition techniques that identify shape, color, or size, may not be able to identify single instances, but only object classes.

Radio frequency identification (RFID) promises to be an unobtrusive, practical, cheap, yet flexible technology for identification of individual instances. There is a wide variety of products and technologies available; the book [1] provides a good overview.

Research efforts are under way to develop radio frequency tags that are either small enough to be embedded even into paper in an unobtrusive way, or cheap enough to

be attached to large quantities of inexpensive goods. The μ-chip by Hitachi [14] is an example of a tiny RFID chip that can be worked into thin materials; sample applications would be paper-based document management and additional security features for bank notes. Another research project, at MIT's Auto-ID Center, aims at developing a very cheap RFID chip, primarily for enhancing supply chain management processes. [1]

In most applications today, typically a single RFID tag is recognized at a time. In electronic article surveillance, for example, it is sufficient to recognize only one unpaid item in order to take appropriate measures. In many other applications, objects are presented sequentially to the reader device, e.g. on a conveyor belt, thus making it unnecessary to recognize more than one item at a time. This allows for very fast object identification.

The ability to recognize many tags simultaneously is crucial for more advanced applications, however. As examples, consider laundry services, warehouses, or the supermarket checkout. We have implemented two applications that use multiple tag identification. The first one is a monitor for card games where cards put on a table are identified in order to keep track of the game's course [10]. The second application is the "RFID Chef"[2], a kitchen assistant that recognizes food items, e.g. in a shopping bag, and makes suggestions about possible dishes involving these items [6]. It is designed to also take the abilities of the cook into consideration when the cook identifies himself with his own RFID tag. Many RFID tags are presented simultaneously to the reader device in both applications and it is crucial to reliably identify all of them.

We have used a commercially available RFID system, "I-Code" by Philips Semiconductors[3]. This system provides the feature of scanning multiple tags simultaneously employing a stochastic anti-collision scheme. (There are other products that advertise multiple tag identification, e.g. TIRIS by Texas Instruments[4].) The communications protocol between the reader and the tags uses a scheme similar to slotted Aloha where slots are provided for the tags to send their messages. Due to physical constraints, tags are unaware of each other and thus, collisions may occur when multiple tags use the same slot for sending. Since tags choose their slots randomly, collisions may be resolved in subsequent read iterations, and after a number of iterations, identification data from all tags can be retrieved.

In this paper we show how to identify a set of tags in such a system if their number is not known in advance. Multiple read cycles are performed until all tags are identified with a given level of assurance. The number of present tags is estimated in each step and the reading parameters are adjusted accordingly. This yields an efficient and reliable procedure for object identification.

The next section summarizes and discusses features of the I-Code system. Section 3 reviews the mathematical tools used for the analysis of the system and the design of the identification procedure. Section 4 presents the results of the analysis, which are exploited in Section 5 that describes an adaptive technique for multiple tag identification

[1] http://www.autoidcenter.org/

[2] http://www.inf.ethz.ch/vs/res/proj/rfidchef/

[3] http://www.semiconductors.philips.com/markets/identification/products/icode/

[4] http://www.ti.com/tiris/

and presents experimental results. The remaining sections deal with related work and present a summary.

2 The I-Code RFID System

A working configuration of an I-Code system consists of the following parts. A so-called "reader" unit is attached to the serial interface of a host (usually a PC), and an antenna is attached to the reader unit by a coaxial cable. The reader unit controls the power that is transmitted by the antenna and encodes the data exchanged between the host and the tags. The programming interface of the reader consists of a set of commands that are described below.

Communication and power transmission between the reader and the tags takes place by inductive coupling between the coil of the reader and the coils of the tags. The channel from the reader to the tags is suitable only for broadcasting and normally all tags within reading range will answer requests from the reader. There are, however, means to address specific tags by "muting" all others. On the other hand, messages sent by tags will only reach the reader device; tags are not aware of each other and cannot exchange messages directly.

The sizes of the coils (of both the reader and the tags) determine the range in which communication can take place. Mid-range antennas with a diameter of approx. 50 cm, as used in our experimental settings, can recognize tags within approx. 50 cm range. The tags we were using were about 5×8 cm^2, so-called "smart labels" attached to an adhesive foil. The tags consist of a chip attached to a printed coil antenna.

The full technical documentation of the system is available from the Philips Semiconductors Web site. Some aspects are reviewed here since they are important for the rest of the paper.

2.1 Tag Memory

An I-Code tag provides 64 bytes memory that are addressable in blocks of 4 bytes. All blocks can be read from, but writing to some blocks is inhibited, indicated by a set of write protection bits. This prevents changes to the serial number and similar data. The write protection bits themselves cannot be deactivated after activation.

Of the 64 bytes, 46 are available for application data. The rest is reserved for a 8 byte serial number and the following functionality: write protection; one bit for indicating electronic article surveillance; one bit indicating the "quiet" state of the tag. If the latter bit is set, the tag will not engage in communication with the reader unless a "reset quiet bit" procedure is executed.

2.2 Programming Interface

The programmatic interface of the system is provided by the reader device. It comprises commands for setting configuration parameters of the reader device itself, e.g. the speed of the serial connection, and commands for handling communication with tags that are in range. Communication commands include the following:

– *Anti-collision/select (ACS).* This command causes all tags that are in range to send their serial numbers. Afterwards, these tags become "selected" and keep quiet in following ACS cycles as long as they are in range. After a tag moves out of the field it becomes "unselected". When it comes back again, it re-sends its serial number. This command can be used to detect tags that are in range, since a list of serial numbers is returned. It is also a prerequisite for writing to tags, since the write command affects only selected tags. However, we are not going to use this command since one ACS cycle takes significantly longer than a *Read unselected* command.
– *Write.* This command is used to write data to a number of tags. One data block (4 bytes) can be written to at a time, but multiple tags may be affected. The tags are selected by the time slot (discussed in the next subsection) they have used while the *ACS* command. This requires that tags don't move in and out of range while writing is in progress.
– *Read.* This command causes only "selected" tags to send their data. It is performed after an *ACS* command.
– *Read unselected.* This is similar to *read* but all tags are triggered regardless of their selection status. By specifying the blocks 0 and 1 to be read, this command can be used to read the serial numbers as well. This is our preferred reading command.

2.3 Framed Slotted Aloha Medium Access

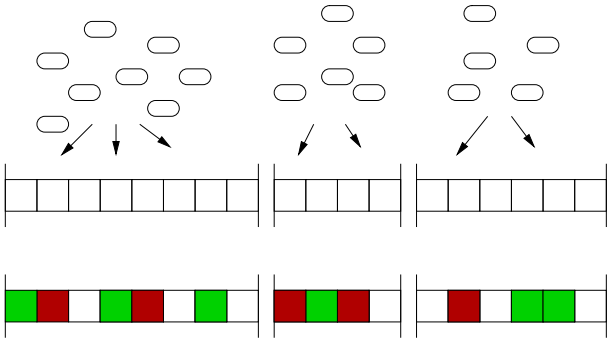

Fig. 1. Tags are randomly allocated to slots within a frame (above). This results in some slots remaining empty, and others containing one or more tags (below). The latter case results in a collision, and no data can be retrieved from these tags

The I-Code system employs a variant of slotted Aloha for access to the shared communication medium, known as *framed* Aloha [11]. After the reader has sent its request to the tags, it waits a certain amount of time for their answers. This time frame is divided into a number of slots that can be occupied by tags and used for sending their answers. When multiple tags use the same slot, a collision occurs and data gets lost (Fig. 1). The

reader can vary the frame size, e.g. for maximizing throughput; the actual size of a slot is chosen according to the amount of data requested.

A tag reading cycle consists of two steps:

1. Reader $\Leftarrow$: *I, rnd, N*
2. $T \rightarrow_{s_T(N,rnd)}$ Reader: $data_{T,I}$ for all tags T

In the first step, the reader device broadcasts a request for data. I denotes what data is requested by specifying an interval of the available 64 bytes of tag memory; $rnd \in [0, 31]$ is a random value whose use is explained below; $N \in \{1, 4, 8, 16, 32, 64, 128, 256\}$ is the frame size and denotes the number of available slots for responses.

In the second step, tags that are in the proximity of the antenna respond ($\rightarrow_s$ denotes a tag sending in slot s, $0 \leq s < N$). A tag T uses a tag-specific function s_T to compute its response slot number, using the frame size and the random value as parameters; the random value is supposed to avoid the same collisions occurring repeatedly. However, we found that this function is to some degree indeterministic. Generally, collision patterns will differ even if the same parameters are provided.

For the purpose of analysis, we are not interested in the actual data returned by the tags. We therefore view the result of a read cycle as a triple of numbers $\langle c_0, c_1, c_\kappa \rangle$ that quantify the empty slots, slots filled with one tag, and slots with collisions, respectively.

We will not take into consideration the *capture effect* by which a tag's data may be able to be recognized by the antenna despite of a collision. The capture effect is quite common if tags are placed close to each other. This means practically that data, which would normally be lost due to the occurring collision, can be read, and thus the system performance rises. However, it seems that whenever such a "weak" collision between the same two tags occurs, one of them always "wins" and the data from the other one is lost. Therefore, the influence of the capture effect is only minimal and seems not to have great impact on the performance.

3 Mathematical Preliminaries

This section reviews some mathematical tools we will use in subsequent sections. The number of slots in a time frame available for tag messages is called "frame size" and will be denoted by N. The number of tags is usually denoted by n.

3.1 Occupancy Problems

The allocation of tags to slots within a time frame belongs to a class of problems that are known as occupancy problems, which are well-studied in the literature [4, 5] and widely applied [8, 9]. These problems deal with the random allocation of balls to a number of bins where one is, e.g., interested in the number of filled bins. In the following, we will speak of "tags" and "slots" instead of "balls" and "bins".

Given N slots and n tags, the number r of tags in one slot is binomially distributed with parameters n and $\frac{1}{N}$:

$$B_{n,\frac{1}{N}}(r) = \binom{n}{r} \left(\frac{1}{N}\right)^r \left(1 - \frac{1}{N}\right)^{n-r} . \quad (1)$$

The number r of tags in a particular slot is called the *occupancy number* of the slot. The distribution (1) applies to all N slots, thus the expected value of the number of slots with occupancy number r is given by $a_r^{N,n}$ (see also [4, p. 114]):

$$a_r^{N,n} = NB_{n,\frac{1}{N}}(r) = N\binom{n}{r}\left(\frac{1}{N}\right)^r\left(1-\frac{1}{N}\right)^{n-r}. \tag{2}$$

Let us denote by μ_r the random variable that equals the number of slots being filled with exactly r tags, $r = 0, 1, 2, \ldots, n$. The distribution of μ_r depends on the probabilities

$$P(\mu_r = m_r) = \frac{\binom{N}{m_r}\ \prod_{k=0}^{m_r-1}\binom{n-kr}{r}\ G(N-m_r, n-rm_r)}{N^n}, \tag{3}$$

where

$$G(M,m) = M^m + \sum_{k=1}^{\lfloor \frac{m}{r} \rfloor}\left\{(-1)^k \prod_{j=0}^{k-1}\left\{\binom{m-jr}{r}(M-j)\right\}(M-k)^{m-kr}\frac{1}{k!}\right\}. \tag{4}$$

The rationale behind these formulas is the following. Imagine a matrix ν_{ij} with n rows (one for each tag) and N columns (one for each slot). Each allocation of tags to slots corresponds to such a matrix where $\nu_{ij} = 1$ if tag i falls into slot j, and $\nu_{ij} = 0$ otherwise; there are N^n such matrices.

The matrices we are interested in represent allocations where there are exactly m_r slots with r tags in each of them. These are the matrices for which the following condition holds. For m_r columns, $\nu_j = r$ holds, and for the remaining $N - m_r$ columns, $\nu_j \neq r$. There are $\binom{N}{m_r}$ ways of arranging these m_r columns. Each of the m_r columns defines a group of r indistinguishable rows. The first group can be arranged in $\binom{n}{r}$ ways, the second group must be drawn from the remaining columns, etc.

The remaining columns and rows can be arranged in $(N-m_r)^{n-rm_r}$ ways, but we have to be careful not to count the arrangements that include allocations of exactly r tags into a slot, i.e. containing columns for which $\nu_j = r$. Function G computes the number of arrangements we are looking for. It determines the correct value by the principle of inclusion-exclusion. The faculty accounts for the arrangements of the columns for which $\nu_j = r$. From this, the formulas (3) and (4) follow.

3.2 Tag Reading as a Markov Process

Suppose one starts identifying a fixed set of tags, having recognized none yet. In each read cycle, one is able to read data from a number of tags (the ones that don't cancel each other out due to collisions). The probability distribution of this number is given by P, defined above. In each step, a certain fraction of the recognized tags is new, i.e. they haven't been recognized in previous cycles. The number of new tags depends on the number of tags already known.

This process of reading tags can be modeled as a (homogeneous) Markov process $\{X_t\}$, where X_t denotes the number of known (i.e., already identified) tags in

step t. The number of tags known in the next step solely depends on the number of known tags in the current step. The discrete, finite state space of the Markov process is $\{0, 1, \ldots, n\}$. The transition probabilities are given by

$$q_{ij} = \begin{cases} 0 & \text{if } j < i \\ \sum_{r=0}^{i} P(\mu_1 = r) \frac{\binom{i}{r}}{\binom{n}{r}} & \text{if } j = i \\ \sum_{r=j-i}^{n} P(\mu_1 = r) \frac{\binom{n-i}{j-i}\binom{i}{r-j+i}}{\binom{n}{r}} & \text{if } j > i \end{cases} . \tag{5}$$

The first case is simple: we cannot go to a state where less tags are known than before, therefore the probability for such transitions is zero. The second case accounts for the possibility of recognizing only such tags that are already known, i.e. the recognized tags in that step are all drawn from the set of already known tags.

In the third case, we recognize exactly $j - i$ tags we do not already know (and possibly some we already do know). That means, we draw $j - i$ tags from the yet unknown $n - i$ tags (and possibly some from the i already known tags).

We will use the matrix $Q = (q_{ij})$ to compute a lower bound of the number of reading steps necessary to identify all tags with a given probability. The initial distribution $q(0)$ reflects the fact that in the beginning we are sure that no tags are known, and is given by

$$q(0) = (P(X_0 = 0), P(X_0 = 1), \ldots, P(X_0 = n)) = (1, 0, \ldots, 0) . \tag{6}$$

3.3 Parameter Estimation

In order to pick the appropriate frame size N for the (a priori unknown) number of tags n in the field, we have to estimate n. Based on the the results of read cycles $c = \langle c_0, c_1, c_\kappa \rangle$ (denoting the number of empty, filled, and collision slots), and the current value of N, we will define functions that compute estimations of n:

$$\mathcal{E} : N, c \mapsto n_\varepsilon . \tag{7}$$

The expected error ε of an estimation function $\mathcal{E}$ tells us something about the quality of the estimate. We use the following error function, which sums up the weighted errors over all possible outcomes of the read cycle:

$$\varepsilon = \sum_c |\mathcal{E}(N, c) - n| P(\mu = c) . \tag{8}$$

4 Identification Performance

For applications like a supermarket checkout or the RFID Chef scenario described in the introduction, where the number of tags is not known in advance, it is not clear how many read cycles have to be performed until the tags are identified with sufficient accuracy. If too many cycles are performed, the delay will be high, inducing cost and worsening the user's experience. On the other hand, some tags might be missed if too few cycles

Table 1. Execution time for *read unselected*. Figures shown are for a 57600 baud connection over RS-232

N slots	1	4	8	16	32	64	128	256
t_N (ms)	56	71	90	128	207	364	676	1304
σ	4.96	2.19	2.26	3.80	4.79	4.82	5.05	4.36
t_N/t_{N-1}	–	1.3	1.3	1.4	1.6	1.8	1.9	1.9

are performed. Therefore, an "optimal" value for the number of cycles should be used, minimizing the required time while maintaining high accuracy. This value, however, varies with the frame size N and the actual number of tags n.

If the frame size is small but the number of tags is large, many collisions will occur and the fraction of identified tags will degrade. Therefore, one might choose to use large frames, but then, response time is always high, even if there are only few tags in range. The choice of large frames also poses a problem in highly dynamic applications where tags leave the range of the reader quickly after entering it: tags that enter the field after the initial request has been sent by the reader will not send an answer.

In this section, we show how to compute the parameters (frame size, number of read cycles) for optimal tag identification w.r.t. to the time required to identify all tags under a given assurance level. The assurance level is given as a probability of identifying all present tags.

4.1 Full Tag Set Identification

Due to the stochastic nature of the reading process, we cannot expect to identify all tags with complete certainty, but we can reach for higher assurance if we are willing to perform more read cycles and wait for their completion. We define the *assurance level* as the probability α of identifying all tags in the field. The desired level of assurance depends on the requirements of our application. We will give figures mainly for $\alpha = 0.99$, which allows for one or more tags missing in less than 1 % of all runs. Note that, the number of tags missed, if any, will be typically very small, thus leading to a high overall recognition rate.

In order to compute the time required to achieve a given assurance level, we need to take into consideration the time requirements for single read cycles. Table 1 shows the cycle time t_N for all possible settings of N in the considered RFID system. The values were obtained by performing read cycles for one minute and computing the average consumed time. The variation of t_N is rather low, as the low standard deviation σ shows. Note that t_N is nearly linear in N as we would expect. Note also that t_N depends on the connection speed between the reader device and the host.

For a fixed frame size N, the time T_α required to achieve an assurance level α is given by

$$T_\alpha = s_0 \cdot t_N \ , \tag{9}$$

where s_0 is the minimum number of read cycles required to identify all n tags in the field with probability α. s_0 is therefore the minimum value of s for which the following

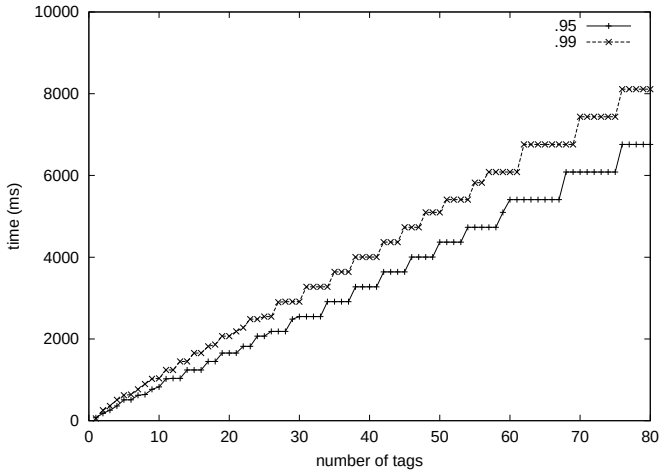

Fig. 2. Time requirement T_α (for optimal N) for $\alpha = 0.95$ and $\alpha = 0.99$

condition holds:

$$Q^s q(0)[n] \geq \alpha \; . \tag{10}$$

Note that the resulting vector of the product $Q^s q(0)$ contains the probabilities of identifying k tags after s read cycles, $k = 0, 1, \ldots, n$. We choose its nth component and compare it to α.

We can now compute the optimal frame size N for a given number of tags n under a desired assurance level. We have performed this computation for $\alpha = 0.95$ and $\alpha = 0.99$. Figure 2 shows the resulting time requirements for optimal choices of N for up to 80 tags. What can be seen is that, the time required increases linearly with the number of tags, and it takes approximately 3 seconds to identify a full set of 30 tags with high probability—if the optimal frame size is known, e.g. if n can be estimated correctly.

4.2 Dynamic Slot Allocation

Figure 3 shows optimal frame sizes and the respective numbers of cycles to perform in order to achieve $\alpha = 0.99$ for up to 160 tags. From this graph we can obtain the optimal value for N for a given number of tags n. (The number of cycles multiplied by t_N from Table 1 yields the graph in Figure 2.) However, in practice it is reasonable to assume that n is not known and has to be estimated based on observed read results. For a read result $c = \langle c_0, c_1, c_\kappa \rangle$ (and the current setting of N), we give two estimation functions that yield approximations for n.

The first estimation function is obtained through the observation that a collision involves at least two different tags. Therefore a lower bound on the value of n can be obtained by the simple estimation function $\mathcal{E}_{lb}$, which is defined according to the template (7) as

$$\mathcal{E}_{lb}(N, c_0, c_1, c_\kappa) = c_1 + 2c_\kappa \; . \tag{11}$$

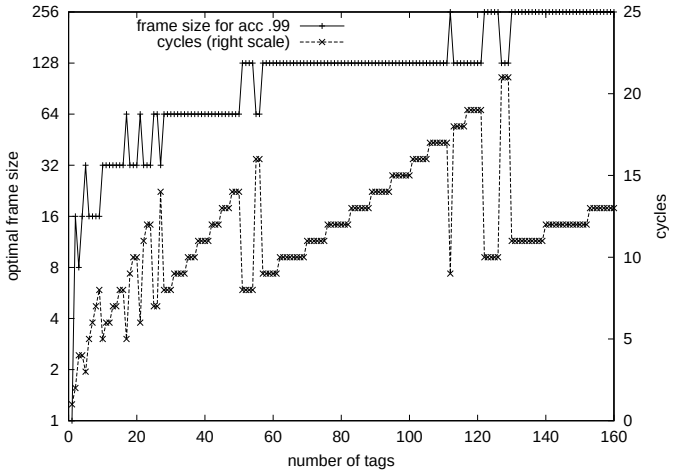

Fig. 3. Optimal frame sizes and numbers of read cycles to perform

A different estimation function is obtained as follows. Chebyshev's inequality tells us that the outcome of a random experiment involving a random variable X is most likely somewhere near the expected value of X. Thus, an alternative estimation function uses the distance between the read result c and the expected value vector to determine the value of n for which the distance becomes minimal. We denote this estimation function by $\mathcal{E}_{vd}$; it is defined as

$$\mathcal{E}_{vd}(N, c_0, c_1, c_\kappa) = \min_n \left| \begin{pmatrix} a_0^{N,n} \\ a_1^{N,n} \\ a_{\geq 2}^{N,n} \end{pmatrix} - \begin{pmatrix} c_0 \\ c_1 \\ c_\kappa \end{pmatrix} \right| . \tag{12}$$

In order to be able to assess an estimation function, we would like to know the expected error of the estimate. This error can be computed by applying the error function given in equation (8). Note that for $\mathcal{E}_{lb}$, the error always states an *underestimation* of the real value of n, which is not the case for $\mathcal{E}_{vd}$.

Approximations of the error of $\mathcal{E}_{lb}$ and $\mathcal{E}_{vd}$, obtained by exhaustive search of the space of possible read results, are shown in Figure 4. What can be drawn from the graph is the fact that $\mathcal{E}_{lb}$ is more accurate for low values of n, while $\mathcal{E}_{vd}$ is more steady for a wider range of n.

What is critical about $\mathcal{E}_{lb}$ is the fact that the error starts to exceed the error of $\mathcal{E}_{vd}$ and becomes quite large just in the range of the transition of $N = 32$ to $N = 64$ as the optimal frame size. Since $\mathcal{E}_{lb}$ always yields an underestimation, this makes the transition more likely to be missed for values $n \approx 30$. Thus, for such n, the accuracy will likely decline.

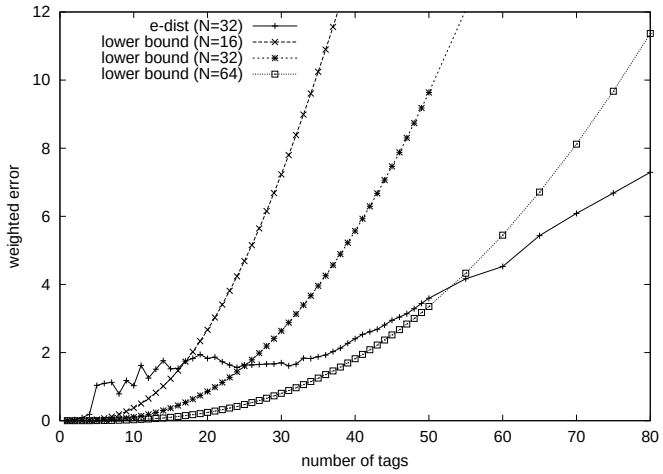

Fig. 4. Error of parameter estimation. The weighted error is the variance of the tag number estimate. $\mathcal{E}_{lb}$ ("lower-bound") is quite accurate for small n but grows fast with larger n, while $\mathcal{E}_{vd}$ ("e-dist") is more steady

5 Adaptive Tag Reading

5.1 Choosing an Optimal Frame Size

Due to the inaccuracy of the estimation functions and the jitter as shown in Figure 3, we are—to some degree—free to choose the actual frame size for a given estimate. For example, if $n \in [17, 27]$, both 32 and 64 are appropriate choices for N, since both settings yield similar times. The intervals [*low*,*high*] for which a certain choice of N is applicable are summarized in Table 2. The table lookup can be implemented in a way that is shown in Fig. 5. This implementation avoids jitter in the result by making conservative transitions between interval borders.

Table 2. Optimality intervals for frame sizes

N slots	1	4	8	16	32	64	128	256
low	–	–	–	1	10	17	51	112
high	–	–	–	9	27	56	129	∞

5.2 Continuous and Static Tag Reading

We can differentiate between two basic scenarios for tag identification. One scenario is static, i.e. a set of tags enter the field and stay there until all tags are identified (with high

```
int adaptFrameSize(N, n_est) {
    while (n_est < low(I(N))) { N = N/2; }
    while (n_est > high(I(N))) { N = 2*N; }
}
```

Fig. 5. Choosing a frame size

probability). An example is the checkout counter where a shopping bag is put and stays there until the terminal has identified all items. The other scenario is rather dynamic, with tags entering and leaving the field continuously.

In the dynamic case, tag reading proceeds without terminating, and the currently identified tag set is reported continuously. Estimating the number of tags and adapting the frame size is nevertheless necessary in order to maximize the identification rate. However, we will concentrate on the static case where the process of tag reading must come to a halt eventually.

5.3 A Procedure for Static Tag Set Identification

In the static case, the tag reading process is started when the first read result $\langle c_0, c_1, c_\kappa \rangle$ with $c_1 + c_\kappa > 0$ is obtained, i.e. at least one tag has entered the field. The process continues until all tags are identified with assurance level α. Once a good estimate for the number of tags is known, and therefore the optimal frame size N is given (by Table 2), the number of read cycles to perform in order to achieve α can be computed using equation (10).

When the process is started, a value for N has to be chosen, but this starting value will not be optimal in most cases. This also yields a first estimate of n that will be inaccurate and we can only hope that adapting N to the estimate will bring us closer to the true value of n in subsequent read cycles.

Therefore, the first few read cycles will contribute only little information about the tag set. They will only bring us closer to the optimal frame size. Once we have reached the optimal N, we can perform the prescribed number of cycles to achieve the desired level of accuracy. Of course, this introduces a penalty due to the first read cycles consuming time without contributing much.

This idea is captured in the algorithm sketched in Figure 6. The algorithm assumes that the tag set in the field is static. This allows to adapt the frame size `N` and the estimated number of tags `n_est` such that these values are monotonically increasing. This not only guarantees termination of the procedure, but also makes the process robust against too low estimates that might occur due to erroneous read cycles.

The starting value for N is set to 16, which is the lowest reasonable value according to Table 2. Note that tags only send their data if they were present in the field at the beginning of a read cycle. Since it is unlikely that the tags enter the field right at the start of a new cycle, the time for the first cycle will be wasted. By choosing a low starting value for N, we minimize this initial idle time. Another reason for this choice is that we are adapting N only by increasing it. Thus, in order to take full advantage of all possible values for N, we have to start with the lowest value.

```
identifyStatic() {
    N = 16; n_est = 0; stepN = 0;
    do {
        stepN++;
        c = performReadCycle(N);
        t = estimateTags(N, c);
        if (t > n_est) {
            n_est = t;
            N0 = adaptFrameSize(N, n_est);
            if (N0 > N) {
                stepN = 0; // restart with new frame size
                N = N0;
            }
        }
    } while (stepN < maxStep(N, n_est));
}
```

Fig. 6. Procedure to adaptively read a static set of tags

The variable `stepN` holds the counter for the cycles performed with the (currently estimated) optimal setting for frame size `N`. When this counter reaches its maximum value, the procedure terminates. The counter is reset to zero whenever a new estimate of `N` is made. Since a new estimate is only accepted if it excels the old one, `N` will eventually reach its maximum and the counter `stepN` will not be reset anymore. The variable `n_est` is more volatile than `N`, but bounded by the actual number n of tags in the field (assuming we employ the estimation function $\mathcal{E}_{lb}$ that always yields a lower bound of n). Thus, termination is guaranteed.

5.4 Experimental Results

We have implemented the tag reading scheme presented here and executed the procedure `identifyStatic` for tag sets of up to 60 tags. The tags were arranged around the antenna of the reader device in a way that we hoped would optimize field coverage. During the test, the tags were not moved. For each tag set, we carried out 100 runs of the identification procedure (without changing the arrangement in between).

The aspired accuracy level of 0.99 (meaning that, out of 100 runs only one run should miss any tags) could not be reached in all tests—the worst level being with a set of 34 tags, where only 92 of the 100 runs yielded the full tag set (see Fig. 7). Not surprisingly, accuracy suffers with increasing tag numbers, due to the fact that it becomes increasingly difficult to arrange a larger number of tags around the antenna while maintaining good coverage. As was mentioned above, we expect the accuracy to drop at around 30 tags due to the increasing error of our estimation function. This would explain the drop that can be actually observed in Figure 7. Another source of inaccuracy are objects located around the testbed (walls, chairs, etc.), which could have a negative influence on the tests. However, the procedure consumed only little time more than

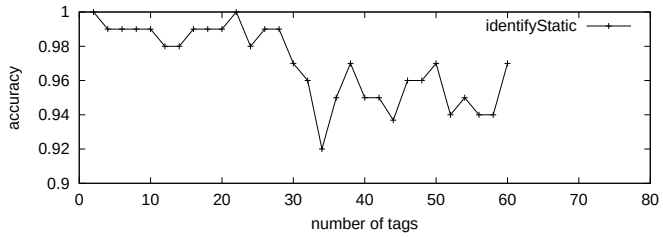

Fig. 7. Accuracy of tag identification in practice. The graph shows the percentage of runs that yielded the full tag set. Not shown is the overall recognition rate (percentage of identified tags over 100 runs), which never dropped below 0.99

would be required if the number of tags were known in advance (see Fig. 8). There are peaks in the zones where the error of the estimate becomes large, which causes good estimates often to be made only after some time has already been spent on a suboptimal frame size. Then, the frame size is adapted, and all cycles are re-performed.

Another figure describing the accuracy of our procedure is the fraction of tags recognized over a large number of runs. In our tests, this figure (not shown graphically) never dropped below 0.99. This means practically that although we might miss some tags in a small number of runs, these misses add up to only a small percentage in the long perspective.

One has to bear in mind that these figures are very sensitive to environmental conditions and the relative positions of the tags to the antenna. We performed our tests in an office environment where we could carefully arrange the tags and were able to somewhat optimize the conditions under which the tests took place. We expect the accuracy that can be reached under real-world conditions to be much lower.

6 Related Work

Apart from mundane applications like inventory and cattle tracking, or improving product management, tagging is often used for attaching information to real objects and for building a bridge between them and their virtual counterparts [2, 15]. Such approaches are based on the assumption that it is often more appropriate to integrate the world of information into our physical environment instead of moving human beings into virtual worlds. This can help to facilitate access to information by means of familiar objects acting as interfaces.

Aloha is a classical communication protocol that is described and analysed in many introductory textbooks. Framed Aloha was introduced in [11, 12]. The underlying model in that work differs from ours in the assumption that nodes are able to detect if a message could be sent successfully and will only re-send it if this was not the case. A procedure for frame size adaptation is given that depends on the outcome of a read cycle and tries to maximize throughput. Frame sizes are, however, not constrained to powers of two. The performance of framed Aloha systems is extensively studied in [16]. The analysis

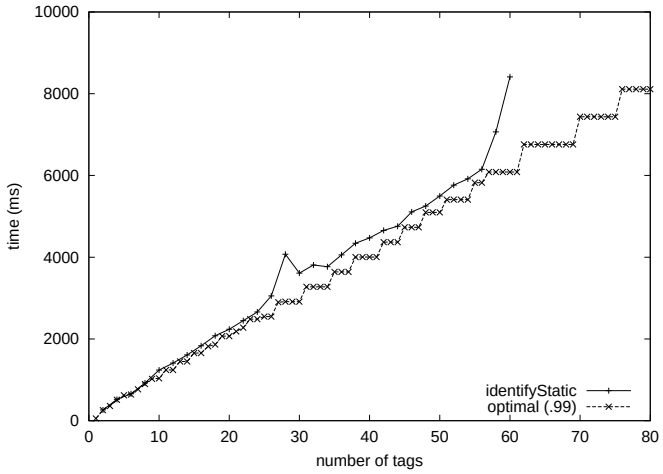

Fig. 8. Actual run time for tag identification. Within the transition areas from one frame size to the next (just below 30 and 60 tags), run time significantly increases. This is due to good estimates arriving late, causing the full number of read cycles to be performed again with a new frame size

takes also into consideration the capture effect, by which a message is received despite of a collision.

Similar to framed Aloha is slotted Aloha with subchannels (S/V-Aloha), which is introduced by and studied with regard to "stability" in [13]. (The system is defined to be stable if the number of blocked users does not exceed a certain point beyond which throughput decreases.) This work uses a combinatorical system model similar to ours.

Some systems employ deterministic anti-collision schemes, e.g. tree-based protocols [3, 7]. Trees are constructed "on the fly" as collisions occur. Each branch corresponds to a partition of the tag set, leafs represent singled-out tags that are identified without a collision. The approaches differ in how the branch of a tag is determined. In [7], ID prefixes determine the partitioning, while in [3], random coin flipping is used. The latter work also considers trees with arity ≥ 2. Both papers investigate how much effort is required for full, reliable tag identification. In contrast to stochastic schemes such as examined in our work, deterministic ones allow for a recognition rate of 100%, but note that this rate is only achievable under optimal conditions; if tags are allowed to enter or leave while the protocol is in progress, accuracy may suffer.

7 Conclusions

We have demonstrated how to efficiently identify a set of RFID tags if the number of tags is not known in advance. We have shown how to determine the parameters for tag reading in order to achieve optimal running time under a given assurance level. The practical implementation we did does not achieve that level, which could be attributed to environmental influences on the experiments. There is also room for improvement of

the frame size adaptation in order to eliminate runtime peaks. It remains challenging, however, to find a procedure that would take environmental effects into account in order to adapt to them. We hope that the work done so far helps implementing pervasive computing environments that employ RFID systems.

References

1. Klaus Finkenzeller. *RFID–Handbuch*. Hanser Fachbuch, 1999. Also available in English as *RFID Handbook: Radio-Frequency Identification Fundamentals and Applications*, John Wiley & Sons, 2000.
2. L. E. Holmquist, J. Redström, and P. Ljungstrand. Token-Based Access to Digital Information. In Hans-W. Gellersen, editor, *Handheld and Ubiquitous Computing*, volume 1707 of *LNCS*, pages 234–245. Springer-Verlag, 1999.
3. Don R. Hush and Cliff Wood. Analysis of Tree Algorithms for RFID Arbitration. In *IEEE International Symposium on Information Theory*, pages 107–. IEEE, 1998.
4. Normal Lloyd Johnson and Samuel Kotz. *Urn Models and Their Applications*. Wiley, 1977.
5. Valentin F. Kolchin, Boris A. Svast'yanov, and Valdimir P. Christyakov. *Random Allocations*. V. H. Winston & Sons, 1978.
6. Marc Langheinrich, Friedemann Mattern, Kay Römer, and Harald Vogt. First Steps Towards an Event-Based Infrastructure for Smart Things. Ubiquitous Computing Workshop (PACT 2000), October 2000.
7. Ching Law, Kayi Lee, and Kai-Yeung Siu. Efficient Memoryless Protocol for Tag Identification. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 75–84. ACM, August 2000.
8. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
9. Fred S. Roberts. *Applied Combinatorics*. Prentice-Hall, 1984.
10. Kay Römer. Smart Playing Cards – A Ubiquitous Computing Game. Workshop on Designing Ubiquitous Computing Games, Ubicomp, 2001.
11. Frits C. Schoute. Control of ALOHA Signalling in a Mobile Radio Trunking System. In *International Conference on Radio Spectrum Conservation Techniques*, pages 38–42. IEE, 1980.
12. Frits C. Schoute. Dynamic Frame Length ALOHA. *IEEE Transactions on Communications*, COM-31(4):565–568, April 1983.
13. Wojciech Szpankowski. Packet Switching in Multiple Radio Channels: Analysis and Stability of a Random Access System. *Computer Networks: The International Journal of Distributed Informatique*, 7(1):17–26, February 1983.
14. K. Takaragi, M. Usami, R. Imura, R. Itsuki, and T. Satoh. An Ultra Small Individual Recognition Security Chip. *IEEE Micro*, 21(6):43–49, 2001.
15. Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. In *Proceeding of the CHI 99 Conference on Human Factors in Computing Systems: the CHI is the Limit*, pages 370–377. ACM Press, 1999.
16. Jeffrey E. Wieselthier, Anthony Ephremides, and Larry A. Michaels. An Exact Analysis and Performance Evaluation of Framed ALOHA with Capture. *IEEE Transactions on Communications*, COM-37, 2:125–137, 1989.