A FRAMEWORK FOR QUICK RFID TAG READDING IN DENSE ENVIRONMENTS

By

MALLA REDDY DEVARAPALLI

Bachelor of Science (Engineering) in Computer Science

Osmania University

Hyderabad, Andhra Pradesh, India

2007

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2007

A FRAMEWORK QUICK RFID TAG READING IN

DENSE ENVIRONMENTS

Thesis Approved:

Dr. Venkatesh Sarangan

Thesis Adviser

Dr. John Chandler

Dr. Nohpill Park

Dr. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

I am indebted to my Graduate Advisor, Dr. Venkatesh Sarangan, and wish to express my sincere gratitude for his constant support, understanding, patient guidance, and encouragement throughout my graduate program. I wish to express my sincere thanks to Dr. John Chandler and Dr. Nophil Park for serving on my Graduate Committee.

I also express my sincere thanks to Food and Agricultural Product Center for supporting me financially to pursue my Masters. I extend my thanks to all the faculty and staff of Computer Science Dept for their constant support.

I also wish to express my sincere thanks and appreciation to my family and friends for their unfailing support and constant encouragement in all my pursuits.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

RFID SYSTEMS

1.1 Introduction

Radio Frequency Identification (RFID) is one of the emerging and useful technologies among the Automatic Identification and Data Capture (AIDC) group of technologies [1]. Technologies in the AIDC collect relatively small amounts of digital data. Barcode technologies, card technologies like magnetic stripes, optical cards, and smart cards belong to AIDC group. RFID has many advantages over the bar codes as it can hold more data and it doesn't require tags to be in line of sight (LOS) with the reader to transfer the data. This technology has been around for many years. A technology which is very much similar to RFID technology was used by allies in the World War II to identify fighter planes as friend or foe [2]. RFID can be used in supply chain management, supermarkets for the speedy checkouts, in identifying the pets, localization of people, libraries etc. Because of its low tag identification rates RFID has not become very popular. For the last few years, researchers have been working in this area and the results are promising. But still there is lot of scope for improvement.

Since RFID has found many applications the systems are becoming denser due to the large number of items being tagged. RFID technology can also be used to track moving objects, hence making the system mobile. We need Medium Access Control protocols which can help RFID systems to function better both incase of dense and mobile environments. The Primary focus of this thesis is to develop an efficient MAC protocol to read RFID passive tags.

This report gives an overview of the RFID technology and also presents careful analysis of various Medium Access Control Protocols that were proposed [1][3][4][5][6]. We have developed a Frame Work which can be used in conjunction with most of the Framed Slotted ALOHA protocols to achieve better average tag reading time, both in static and mobile systems.

1.2 RFID Basics

Radio Frequency Identification Systems uses wireless communication medium to automatically identify the objects. They consist of the components like tags, readers, antenna, and air interface, middleware, central database etc. [7].

- A tag is silicon micro chip which can store few hundred bytes and with a limited memory and computational power. Tags are attached to the objects that need to be identified
- A typical RFID reader is a radio enabled device that interrogates / communicates with the tags to identify them. These readers are of the size of a hand held device connected to a back end data base. The Functions of a RFID reader are to act as power transponder (tag), identifying transponder (tag), read/write information from/to tag, and to interface with a RFID middleware
- Air interface is the transponder reader interface which is used to transmit the radio waves from reader to tag and vice versa. Functions of an air interface are interrogation of tags by creating magnetic or electric field, authenticating the tags, prevention of tag collision, and optimization of power and memory required.

- Middleware is used to convert the data from the tags into a form which is easily understood by the back end ERP systems. A Middleware which acquires data through control signals integrates various business systems.
- A typical back end system is a computer device which is connected to a central database and its functions are to manage acquired data, to generate reports, and to integrate with other business systems.

A typical RFID system is shown below.

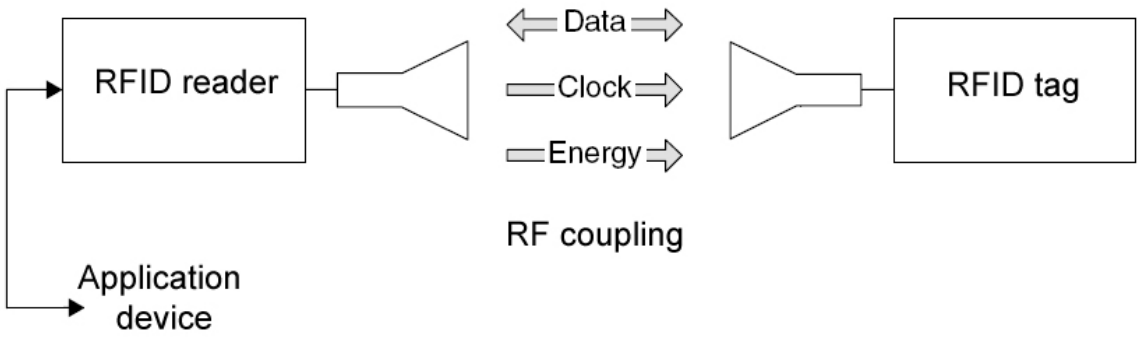

Figure 1.1 Typical RFID System

In typical RFID systems, the reader initiates the communication by powering up the un-powered tags to acquire their identification strings. Readers generally use a continuous radio frequency wave to power the tags [6]. Tags, powered up by the reader's RF waves, respond with appropriate data.

Each tag consists of unique identification string called as Electronic Product Code (EPC) defined by EPC Global [8]. The two types of EPC versions being used are 64 bit and 96 bits memory. A 256 bit version of EPC is being proposed. Basic format of an EPC is given below.

Header	EPC Manager Number	Object Class	Serial Number

Figure1.2 Typical EPC Structure [8]

•Header

Identifies the length, type, structure, version, and generation of the EPC

•EPC Manager Number

Entity or Manufacturer responsible for maintaining the subsequent partitions

•Object Class

Identifies a class of objects (exact type of product, most often the Stock Keeping Unit)

•Serial Number

Identifies the instance of a tag class

Table1.1 shown below lists the various EPC structures being used and proposed. Currently

Table 1.1 Illustrations of various EPC structures [8]

EPC Type	Version size	Domain Manager	Object Class	Serial Number
256- bit Type I	8-bit	32-bit	56-bit	192-bit
256- bit Type II	8-bit	64-bit	56-bit	128-bit
256- bit Type III	8-bit	128-bit	56-bit	64-bit
96-bit	8-bit	28-bit	24-bit	36-bit
64-bit Type I	2-bit	21-bit	17-bit	24-bit
64-bit Type II	2-bit	15-bit	13-bit	34-bit
64-bit Type III	2-bit	26-bit	13-bit	23-bit

1.3 Classification of Tags

A RFID tag is attached or fixed to the object that is being tracked or identified. A tag is a silicon microchip with an antenna that sends data to the RFID reader. A RFID tag contains a unique serial number (EPC) and also some additional data. RFID tags can be active, passive, and semi-passive [9].These categories are briefly explained below.

Passive Tags

Passive tags do not have a battery so they can not initiate the communication by themselves. In this case the reader will initiate the communication process and the tags powered by the reader with a continuous radio signal respond to the reader queries.

Advantages of passive tags are: small size, long operational life, and low cost. Disadvantages are: short reading ranges, need for high powered readers, limited memory. Moreover they are only read-only tags.

Active Tags

Active tags are continuously powered by the battery which is associated with them and are capable of sending more information to the reader. Active tags have a larger range (up to 12m) which makes them more costly. Advantages of active tags are long read ranges, self activation in presence of a reader, and they can be both write/rewrite. In spite of the many advantages, these tags have a few shortcomings like large size, limited operational life, and high installation and maintenance costs.

Semi-passive Tags

Semi–passive tags are similar to passive tags but with a battery powered tag circuitry and more memory gives them the advantage of faster response times and longer read ranges than the passive tags.

1.4 RFID communication model

RFID communication protocol basically consists of three layers [10] and they are briefly discussed below.

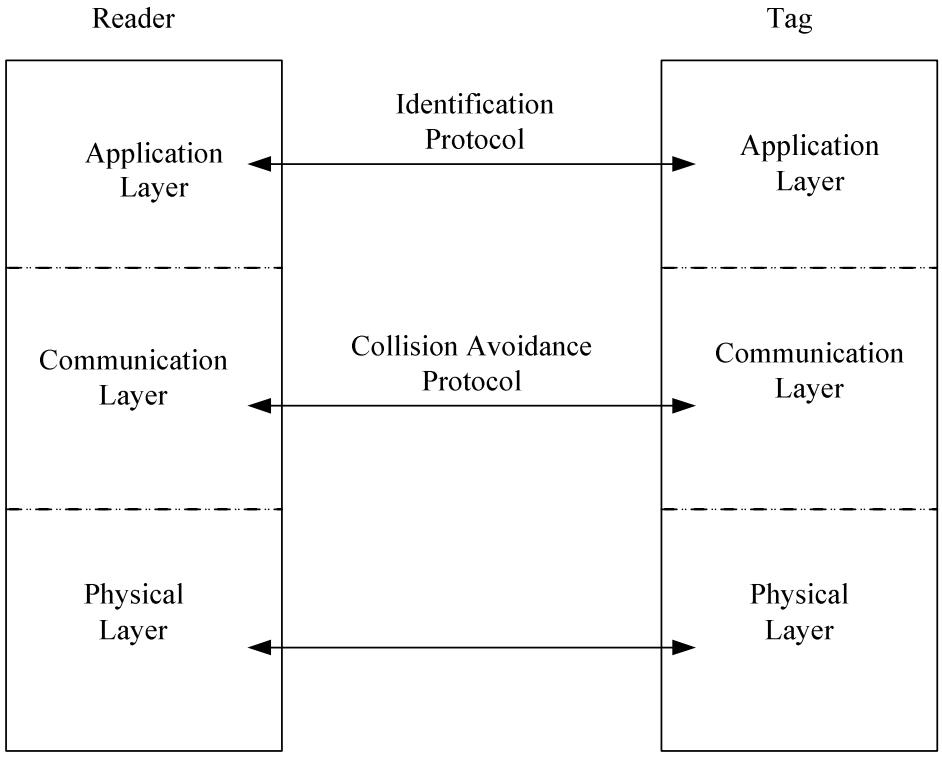

Figure 1.3 RFID Communication Model [10]

Application layer: It consists of an identification protocol(s) to identify tags uniquely.

Communication layer: Since RFID systems use air interface or wireless radio communications to collect the object's information they need to avoid collisions when multiple tags are transmitting their data. In this layer 2 types of collision avoidance protocols (Deterministic and Probabilistic) are in use and they are explained in the following chapter.

Physical layer: This layer uses air interface (frequency, modulation, etc.).

CHAPTER 2

RFID MAC PROTOCOLS

2.1 Introduction

Typically RFID reader queries tags and the tags which are powered by the reader [3], respond to those queries. When multiple tags reply at the same time, this results in a collision. Since the computational power of RFID passive tags is limited, they cannot communicate among themselves to avoid these collisions. So the RFID tag reader has to take care of collisions present during the read process. RFID systems generally use collision avoidance protocols such as ALOHA based algorithms [1] [3] [4] [5] or Tree Search Based algorithms [6] to avoid these collisions.

This Chapter explains various collision avoidance protocols, their advantages and disadvantages in detail.

.

2.2 Framed Slotted ALOHA Algorithms

In Slotted ALOHA algorithm, tags send their serial number to the reader in the slot of a frame and they are identified correctly when only one tag sends its information in that slot. A collision occurs when multiple tags send their data in a single slot. A time slot is a time interval during which tags send out their serial number. The current RFID systems use a type of Slotted ALOHA algorithm known as Framed Slotted ALOHA algorithm. A frame is a time interval between two reader requests consisting of multiple slots [5].

In the following subsections we present various framed slotted ALOHA protocols, their merits and demerits.

2.2.1 Basic Framed Slotted ALOHA

Basic Framed Slotted ALOAH (BFSA) is the first of its kind and uses a fixed frame size. Initially all the tags in the reader's filed are powered and synchronized by continuous RF wave by the reader. In BFSA, the reader broadcasts a frame size to the unidentified tags and waits for their responses [3]. Each tag in the reader's field computes a random number which is less than the given frame size and sends their EPC in those time slots. Multiple tags transmitting their EPC in the same time slot results in a collision. It is likely that some of the time slots are not chosen by any of these tags which are known as idle slots. A reader can successfully read the EPC of a tag in a slot only when no other tag chooses the same slot.

Figure 2-1 explains the reader, tag communication with a fixed frame size of 3 time slots. Initially the reader broadcasts the frame size to all the five tags that are in the reader's region. In the first round of communication, Tags 1 & 3 chose the first slot, Tags 2 & 5 the second slot, and Tag 4 the third slot. The reader detects the collisions in slots 1& 2 and it successfully identifies the Tag4 in slot3.The reader broadcasts an acknowledgement in the form of a bit map to notify the tags whether a particular slot is a success slot or not.

. The tags that communicate in those success slots get muted or killed and the remaining tags participate in the subsequent rounds.

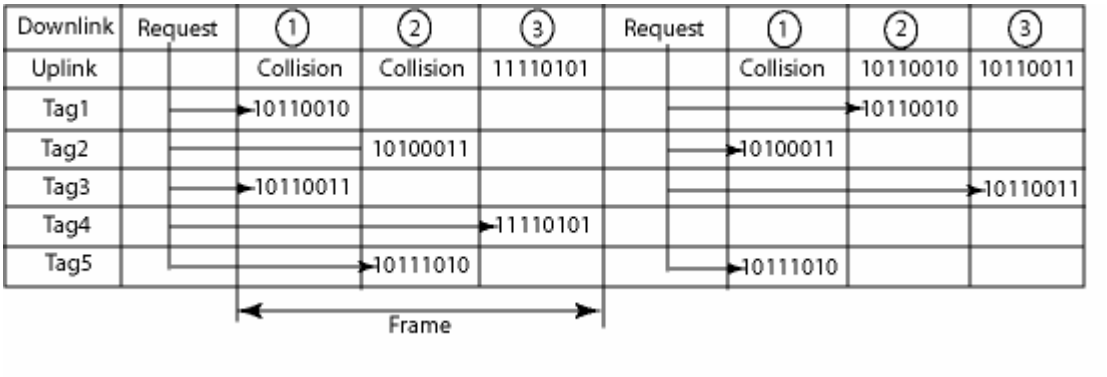

Figure 2.1 Reader-Tag communications in BFSA [5]

Though the protocol is simple to implement it suffers from many disadvantages. Its disadvantages come from its fixed frame size.

- If the broadcasted frame size is smaller than the amount of tags that are participating in the round results in large number of collisions.
- If the frame size is larger than the count of the participating tags, it results in several idle slots.

None of the above cases are desired.

2.2.2 Dynamic Framed Slotted Aloha

The Reader-Tag communication mechanism in Dynamic Framed Slotted ALOHA is similar to that of BFSA, except that it changes the frame size dynamically for efficient tag identification rate. To adjust the frame size, it uses the statistical data collected by the reader, such as number of successful slots, collision slots, and idle slots in the previous round [4]. DFSA has two different types of versions based on the methods of changing the frame size.

The first algorithm [5] changes the frame size based on the number of the idle slots, the slots with collision and the slots filled with one tag. The reader increases the frame size if the number of collision slots is over the upper threshold. If the number of collision slot count is below the lower threshold, the reader decreases the frame size. The reader can identify the tags efficiently in case of a smaller tag count without changing the frame size as it starts with a small frame size. When the number of tags is large, the reader increases its frame size to decrease the collision probability.

The second variant starts a reading process with an initial frame size of two or four. If none of the tags are identified during the previous read cycle, the algorithm increases the frame size and starts another read cycle. It repeats the above process until at least one tag is successfully identified. The reader stops the current read cycle immediately after identifying a single slot. It starts another read cycle with the initial minimum frame size to read another tag [5].

Dynamic Frame Slotted ALOHA works better than its predecessor (BFSA), as it regulates the frame size accordingly. It works better if the number of tags to be read is smaller but in case of a larger tag count, increasing the frame size alone may not be the best solution because it can not increase the frame size indefinitely [11].

2.2.3 Enhanced Frame Slotted Aloha

As with other DFSA protocols, EDFSA collects information such as, the number of successful slots, collision slots, and idle slots - to estimate the number of tags to be read. In case of smaller tag counts, it adjusts the frame size according to the population. For large tag counts, it does not increase the frame size indefinitely. Instead, it restricts

the number of tags that can participate in a given round in order to achieve better reading rate [5].

After estimating the total number of unidentified tags, EDFSA logically divides them into various groups. The number of groups depends on the maximum frame size and the tag population. If N is the maximum frame size and K is the number of unread tags estimated, the tag population is divided into $M = \left\lceil \frac{K}{N} \right\rceil$ groups [5]. The reader then broadcasts this group number along with the maximum frame size. The tags then carry out modulo operation on a random number that they generate with the group size advertised by the reader. The tags with a zero remainder participate in that round. The tags generate a second random number to decide the slot number during which they transmit their data. The frame sizes for different tag counts are estimated such that the overall system efficiency hovers around 36.8% [5].

Assuming a maximum possible frame size of 256, the following table gives a list of the appropriate frame sizes and the number of groups for various unidentified tag populations [5].

Table 2-1 gives the optimal frame size and the modulus for each set of unidentified tag count.

Table 2.1 Number of unread tags vs. optimal frame size and modulo [5]

Number of unread tags	Frame Size	Modulo (M)
: :	: :	: :
1417-2831	256	8
708-1416	256	4
355-707	256	2
177-354	256	1
82-176	128	1
41-81	64	1
20-40	32	1
12-19	16	1
6-11	8	1
: :	: :	: :

2.2.4 Adaptive Slotted ALOHA Protocol

Adaptive Slotted ALOHA protocol offers an optimum frame size for each round using the estimated tag count in the reader's region. ASAP has different times for unoccupied (collision or successful), $T_{B,\ and}$ for idle slots, T_I ($T_B \neq T_I$) [1]. It closes the idle slots prematurely to avoid the wastage in time and bandwidth due to them.

Equation 2.2.4 -1 gives the efficiency function for the ASAP protocol [1].

$$Peff = \frac{E[S].T_B}{E[U].T_B + E[I].T_I} \quad (2.2.4\text{-}1)$$

Where E[S], E [U], and E [I] are expected number of successful slots, unsuccessful slots, idle slots respectively. Given the estimated tag count, k, it proposes value for N (frame size) to maximize the P*eff*.

Using the binomial distribution [1],

$$\text{E [S]} = k(1-1/N)^{k-1} \quad (2.2.4\text{-}2)$$

$$\text{E [I]} = N(1-1/N)^{k} \quad (2.2.4\text{-}3)$$

$$\text{E [U]} = N - k(1-1/N)^{k-1} - N(1-1/N)^{k} \quad (2.2.4\text{-}4)$$

Substituting (2.2.4-2), (2.2.4-3), (2.2.4-4) in (2.2.4-1) gives

$$\text{P}eff = \frac{1}{\beta e^{1/\beta} + (\alpha - 1)\beta} \quad (2.2.4\text{-}5)$$

Where α = TI/TB, and N= β k (β = 1.943)

ASAP uses a variable frame size N= β k, where k is estimated using exp estimation. The estimated tag count using exp estimation is give by

$$\text{K exp} = \frac{\log(\frac{Z_I}{N})}{\log(1-\frac{1}{N})} \quad (2.2.4\text{-}6)$$

Where E [I] ≈ Z_I

Though ASAP claims the better average tag reading time than any of the other protocols, it uses unlimited frame sizes- which is not acceptable.

2.3 Binary Search Tree Protocol

In this section we present the "reader talks first" anti collision binary search tree protocols. The tags respond back to the readers only after the reader requests. This protocol uses contention resolving and collision free methods to identify the tags. Amplitude-Modulated (AM) carrier achieves the reader to tag communication and the tag to reader communication is done by passive back scatter carrier [6]. Tags that are in the reader region can be represented as binary tree with a null root. Each path leading from root to a leaf node represents an EPC of a tag in the reader region. Figure 2.2 represents such a tree. In this model Most Significant Bits (MSB) are at the top and Least Significant Bits (LSB) are at the leaf level.

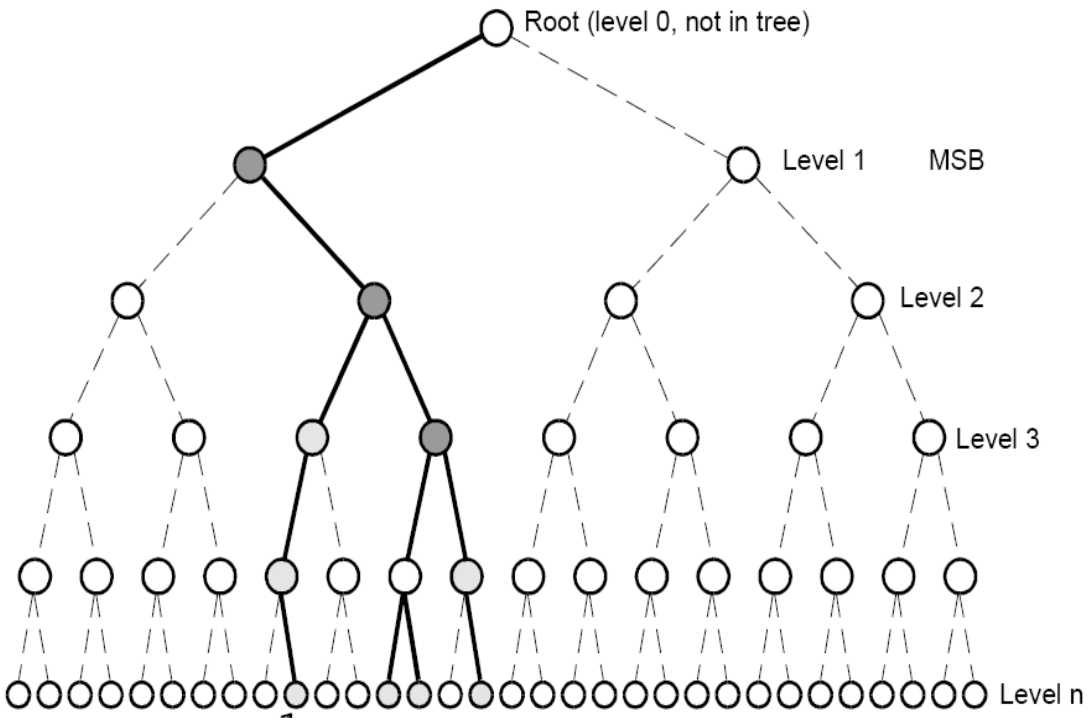

Figure 2.2 Tags EPC as a tree [6]

The protocol reads each tag on a bit-by-bit basis as information is progressively received. Each tag uses bit 0 or bit 1 to inform its response. Initially, reader broadcasts a null and all the tags in the region will respond with their MSB bits. Reader observes a collision if tags have different MSB bits. Reader chooses between one of the two options available and broadcasts it. Tags send their next most significant bit if the broadcasted bit matches with their last backscattered bits otherwise they go to a temporary inactive state. The process repeats until the reader successful reads the complete EPC of a tag [6].

The reader repeats the whole process once it successfully identifies a tag.

CHAPTER 3

PROPOSED FRAME WORK

3.1 System Model

In this report, we consider the collision limited 900 MHZ UHF RFID system where the large number of tags communicates with the reader over a shared channel. Each tag transmits its 64 bit identifier (**E**lectronic **P**roduct **C**ode) and 16 bit CRC with symbol duration of $4\,\mu s$ [6]. Tags employ FSK backscatter modulation to transmit data bits. Reader uses symbols '0', '1', and Null to communicate with tags. The duration of these symbols is $12\ \mu s$ [6]. Reader uses 900 MHz continuous wave to power the tags.

3.2 Accelerated Framed Slotted ALOHA (AFSA)

In this section we propose a frame work which has five different phases to improve the overall average tag reading time. We use Enhanced Dynamic Frame Slotted ALOHA to implement the proposed framework and we call the resulting protocol Accelerated Framed Slotted ALOHA (AFSA). To begin with, the reader sends a continuous radio signal to calibrate the tags and to energize them. Under the proposed framework, five phases are used to complete a single round of tag reading. The different phases are described below.

3.2.1 Advertisement Phase

During this phase, the reader broadcasts the frame size (N) and the number of groups (M) to all the tags that are within its range. As said earlier, upon receiving the values for

N and *M*, the tags generate two random numbers. They perform modulo operation on the first random number with the number of groups in order to identify if they could participate in this round. They are allowed to participate only if they have a zero remainder. If a tag is allowed to participate, it generates a second random number using a uniform distribution in the range of 0 to *N* in order to identify the slot during which they can transmit their data.

Reader Command (8bits)	Frame Size(8bits)	Number of groups $\log_2$M bits

Figure 3.1 Typical structure of reader-to-tags broadcast message

Figure 3-1 shows the typical the typical structure of a reader-to-tags broadcast message. The first 8 bits are used classify the reader command type, second 8 bits depict the frame size and remaining bits ($\log_2$M) tells the number of groups or modulus operator value.

3.2.2 Reservation Phase

This is the second phase, during which the tags let the reader know which slots they have chosen. Unlike other slotted aloha protocols, where in the tags send their data right away in the chosen slot, in the proposed framework, the tags transmit an *n* bit sequence in their chosen slot, where *n* is a protocol parameter. For a given value of *n*, there are 2^n possible *n* bit sequences. A tag randomly picks one of these 2^n sequences and transmits it to the reader in the slot chosen by it.

n bits	n bits	n bits	n bits	n bits	……

Figure 3.2 Tags response to the reader's broadcast message

If the reader successfully receives this *n* bit sequence in the slot chosen by the tag, then it implies that the tag has successfully reserved this slot for transmitting its data. Since each tag transmits only an *n* bit sequence in a slot, the total duration of this phase will be *N***n* bit times.

3.2.3 Reservation Summary Phase

During this phase, the reader lets the tags know the status of their reservations. The reservation status is advertised through a bitmap of length *N.* For example, assuming $N = 4$, the bitmap 1001 indicates that an *n* bit sequence was successfully received by the reader in the slots numbered 1 and 4. This indicates that those tags that had chosen slots 1 and 4 were successful in their reservation. In other words, occurrence of bit 1 in location *i* indicates that only one tag had chosen slot *i*. During slots 2 and 3, the reader did not receive any n bit sequences. This can indicate one of the two possibilities: (i) collisions – these slots were indeed chosen by more than one tag and when the tags transmitted their randomly chosen n bit sequences, collision occurred and consequently, the reader was not able to decode the received signal; or (ii) idle – the slots were not chosen by any of the tags. In other words, occurrence of bit 0 in location *i* indicate that slot *i* has been wasted either due to collisions or idleness.

Reader Command (8 bits)	10111000111110000011111011101111……

Figure 3.3 Reservation summary (bit map)

Thus the bitmap summarizes the status of the reservations made by the tags. It is easy to see that the duration of this phase will be *N* bit times. Figure 3-3 shows a typical structure of the reservation summary. The bitmap of the tags reservation status are followed by the 8 bit reader command.

3.2.3 Data transmission phase

After phase 3, all the tags will be aware of the status of their reservations. Only those tags that find their reservation to be successful will transmit their data in the next phase, namely the data transmission phase. We note that if *S* is the total number of '1's in the N bit summary bit string, it is clear that only *S* data transmissions are possible. In addition, a tag that finds a 1 in the summary bit string in the location corresponding to its chosen slot (say *i*) can easily determine its place in the order of data transmissions by counting the number of '1's in the summary bit string starting from location one until location *i*. Therefore, it is clear that the data transmission phase will have only *S* data transmission slots, where $S <= N$.

EPC + CRC (80 bits)	EPC + CRC (80 bits)	EPC + CRC (80 bits)	EPC + CRC (80 bits)	EPC + CRC (80 bits)	……

Figure 3.4 Tags EPC transmissions

Figure 3-4 shows the typical structure of the data transmission phase. Tags send their 64 bit EPC and 16 bit CRC (Cyclic Redundancy Check) in their reserved slots.

3.2.4 Acknowledgment phase

This is the final phase in which the reader acknowledges the data transmitted by the tags. The acknowledgment is sent in the form of a bit – 1 indicates that the transmission was received successfully and a 0 indicates that the transmission was not successful.

Reader Command (8bits)	1111111111111111101110111111111111111….

Figure 3.5 Acknowledgement bit map: Reader-Tags

All tags that received a positive acknowledgment from the reader will become muted or killed, and will not participate in the tag reading process again. Tags that received a negative acknowledgment will participate in the reading process again. The duration of this phase is S bit times.

3.3 Remarks

In the proposed framework, the total number of information bits that are exchanged between the reader and the tags is $Nn + N + S.64$. In the original EDFSA protocol, the total number of bits that are exchanged between the reader and tags is $N. 64$. It can be shown that $S < N$. Therefore, if we choose n << 64, the total number of bits exchanged

during a single round in the proposed framework can be made lesser than that of the EDFSA protocol. This in turn can lead to shorter round times, thus decreasing the overall tag reading time.

CHAPTER 4

PROTOCOL OPTIMIZATION

4.1 Tag Estimation

In order to broadcast the appropriate frame size and the group number, the AFSA protocol needs to estimate the number of tags that need to read. In the original EDFSA protocol, the number of unread tags is estimated based on three parameters – number of successful slots in the previous round, number of collision slots[], and number of idle slots. An alternate strategy has been proposed in reference [6], in which the Exp estimation method is used to estimate the number of unread tags based on the number of idle slots observed. Either of the above approaches can be used to estimate the tag count. In our work, we chose the approach presented in [6] on account of its simplicity. Let *k'* be the number of tags estimated by the reader after a round. It is to be noted that in the EDFSA protocol, if the group count $M > 1$, then only a fraction of the unread tags participate in a round. Therefore, the total number of unread tags can be arrived at by multiplying the estimated tag count *k'* with the number of groups.

4.2 Optimal n: n^*

In 3.2.3, we have mentioned that during the reservation phase, each tag transmits a randomly chosen n bit sequence in the slot selected by it. If more than one tag chooses the same slot, the reader can detect this collision, provided the bit sequences transmitted by all the tags that have chosen the same slot are different. However, if the bit sequences transmitted all the tags that have chosen the same slot are the same, the reader cannot detect the collision occurring in that slot, i.e., the collision can go undetected. The

consequence of a collision going undetected is that, the reader may wrongly interpret a slot reservation to be successful when in reality, it is not. This incorrect interpretation will lead to collisions in the data transmission phase. In other words, undetected collisions increase the number of bits transmitted during a round thereby increasing the round time and overall tag reading time.

The probability of a collision going undetected depends on two factors – (a) the probability of more than one tag choosing the same slot and (b) the probability that all the tags that have chosen the same slot select the same n bit sequence. The former is decided by the value of frame length N and the number of tags K competing for the slots, while the latter is decided by the value of the parameter n. Since the values of N and K are decided by EDFSA, the only parameter that can be varied to reduce the number of undetected collisions is n, the length of the bit sequence used by the tags to reserve their slots.

Intuitively, it is clear that by opting for a large value for n, the number of undetected collisions can be almost brought to zero. However, increasing the value of n also implies that the tags have to transmit more number of bits, consequently increasing the round time and overall tag reading time. Thus both smaller and larger values of n increase the round time. We now set out to determine the optimal value, n^*, for n under which the total round time is minimized for given values of N and K.

Let T denote the total duration of a round. Let T_{Ad}, T_R, T_{Su}, T_D, T_{Ack} respectively denote the durations of the advertisement, reservation, summary, data transmission, and the acknowledgment phases. Now, T can be written as

$$T = T_{Ad} + T_R + T_{Su} + T_D + T_{Ack} \quad (4.2\text{-}1)$$

A closer look at the different phases of the proposed framework will reveal that, the value of n affects the values of T_R, T_D, and T_{Ack} alone[1]. Once the durations of reservation, data transmission, and the acknowledgment phases are known in terms of n, the value of n^* can be determined by differentiating T with respect to n and equating the outcome to zero.

As mentioned earlier, the duration of the reservation phase is $12.5*Nn$ µsec, while the duration of the data transmission phase is $S*320$ µsec. The duration of the acknowledgement phase is $S*12.5$ µsec. The parameter S indicates the total number of successful slot reservations as perceived by the reader. It can be written as

$$S = E[R] + E[UC] \quad (4.2\text{-}2)$$

Here *E[R]* represents the average number of slots that were truly reserved (i.e., these are the total number of slots that were each selected by exactly one tag) and *E[UC]* represents the average number of collision slots that were undetected by the reader. Since the tags choose one among the N slots as per a uniform distribution, the value for *E[R]* can be written as [1]:

$$E[R] = K\left(1 - \frac{1}{N}\right)^{K-1} \quad (4.2\text{-}3)$$

[1] The other two phases always have durations that are independent of n. They are dependent only on the values of N and K.

A collision goes undetected when more than one tag chooses the same slot and all such tags select the same bit sequence. Since the slot selection and bit sequence selection are independent of each other, it is possible to write the value of *E[UC]* as:

$$E[UC] = N \sum_{i=2}^{K-E[R]} (\Pr obability\_that\_i\_tags\_in\_the\_same\_slot) \frac{1}{2^{ni-1}} \quad (4.2\text{-}4)$$

While the above expression gives the accurate value of *E[UC]*, it becomes cumbersome to handle its differential when we attempt to find the optimal value for *n*. Therefore, we derive an alternate simpler expression that gives an approximate value of *E[UC]* and use this expression to determine *n**.

Let *E[U]* represent the total number of slots that were chosen by more than one tag. The value for *E[U]* is given by:

$$E[U] = N - E[I] - E[R] \quad (4.2\text{-}5)$$

Where E [I] refers to the total number of slots that were not chosen by any of the tags. Given that each tag independently selects any particular slot with equal probability, *E[I]*can be written as $E[I] = N\left(1 - 1/N\right)^K$ [1]. It is easy to observe that if the reader fails to detect a collision in a particular slot, then that slot should be one of the *E[U]* slots. Also by definition, the *E[U]* slots are chosen by at least two tags (possibly more). Therefore, if P_2 represents the probability of two tags selecting the same *n* bit sequence in the reservation phase, then *E[UC]* can be written as

$$E[UC] \leq E[U] \cdot P_2 \tag{4.2-6}$$

It can also be noted that P_2 is equal to $1/2^n$. We use the above upper bound of *E[UC]* to approximate E[UC]. Since we approximate E[UC] by its upper bound, the resulting value of n* will be more conservative, i.e., it will be greater than or equal to the true value of n*. Combining all of the above, we can write T as

$$\begin{aligned} T = {} & T_{Ad} + 12.5Nn + T_{Su} + 320(E[R] + E[U].P_2) + \\ & 12.5(E[R] + E[U].P_2) \end{aligned} \tag{4.2-7}$$

Minimizing the above expression with respect to *n* gives us the optimal value of *n* which can be written as

$$n^* = 3.32 \log_{10}\left(\frac{19.13E[U]}{N}\right) \tag{4.2-8}$$

Using the above value of n^* in the protocol minimizes the total round time for given values of *N* and *K*.

4.3 Numerical Results and Discussion

In this section, we discuss the various simulation results for the AFSA. All of the simulation results are an average of 25 individual experiments. The simulations are performed using C++ and MATLAB.

4.3.1 Number of undetected collisions in AFSA with varying n

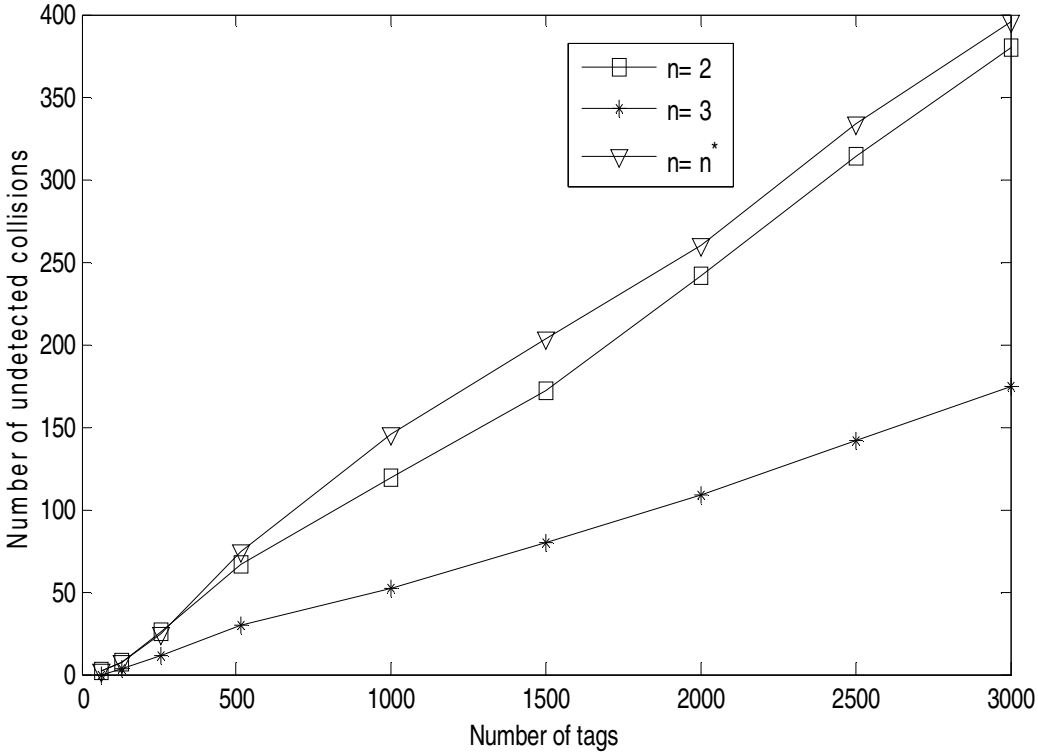

Figure 4.1 Number Undetected Collisions vs. n

During the section 4.2, we have mentioned that some of the collisions go undetected during the reservation phase if tags choose the same n bit sequence and same slot to send its data. Figure 4-1, is a plot for the number undetected collisions for each set of tag count with different n values. It is found that if we use a larger n value then we will have smaller number of collisions which go undetected.

4.3.2 Average tag Reading times for AFSA with different n values

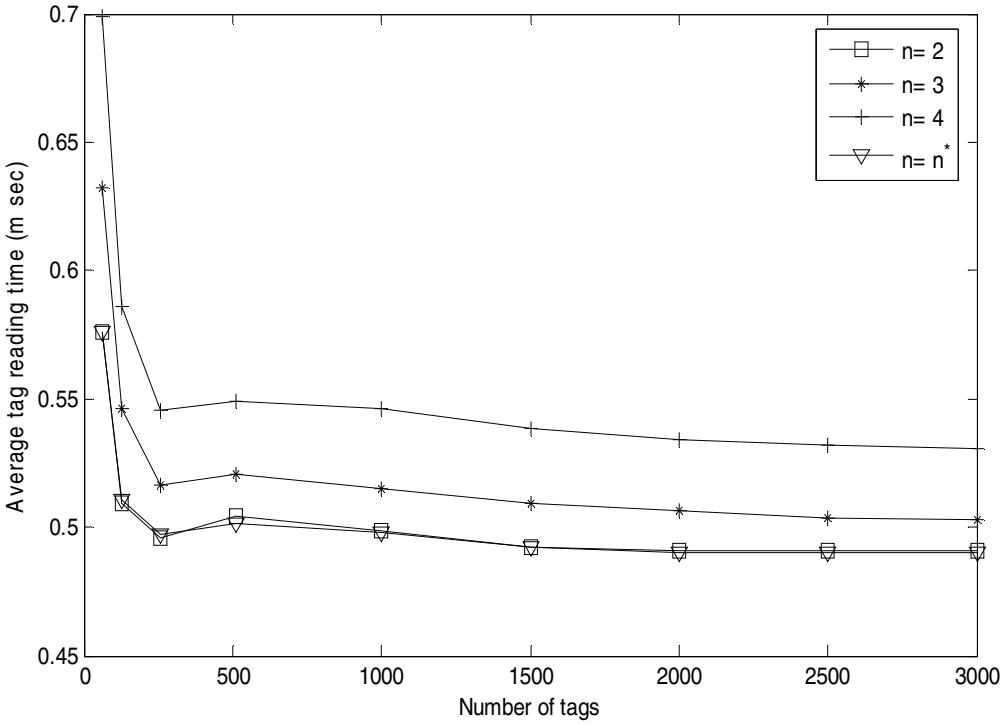

Figure 4.2 Average Tag Reading Time vs. n

Figure 4.2 gives the simulations results of average tag identification time in m sec for various tag counts using different 'n' values. From the figure, If we use n = 2 we can achieve the better average tag reading to that of n = 3 and n= 4. From the figure, we can say that both n* and n = 2 result in similar average tag reading time. This is because the value of n* computed using () is always around 2. Since we cannot broadcast the fraction bit, we have to round the value of n* to a nearest integer.

4.3.3 Comparison of AFSA with EDFSA and ASAP

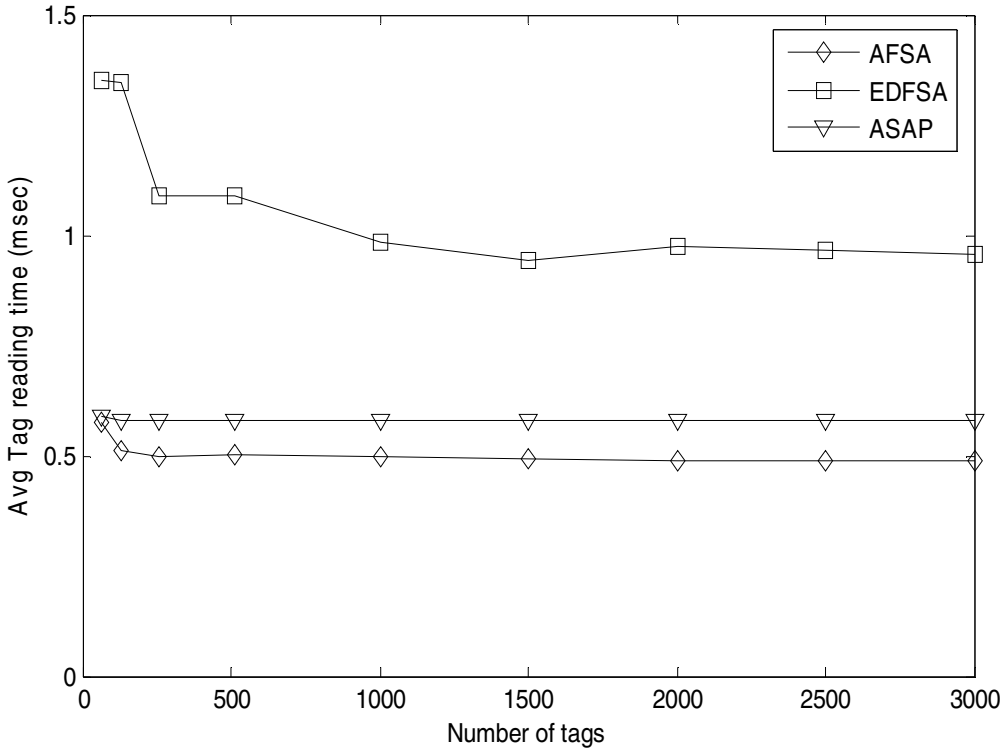

Figure 4.3 Performance comparison: AFSA (n*) vs. EDFSA and ASAP.

Figure 4.3 gives the average tag identification times for AFSA, EDFSA, and ASAP. Most of the time, the average tag reading time for EDFSA is above 1 m sec where as ASAP 0.58 m sec [1]. ASAP's lower tag identification time is due to the fact that it has a variable and unbounded frame size policy. It is found that AFSA shows a steady performance with an average tag identification time of 0.495 m sec.

4.3.4 Statistical deviations

In this section, we present the deviation in the collected statistical data.

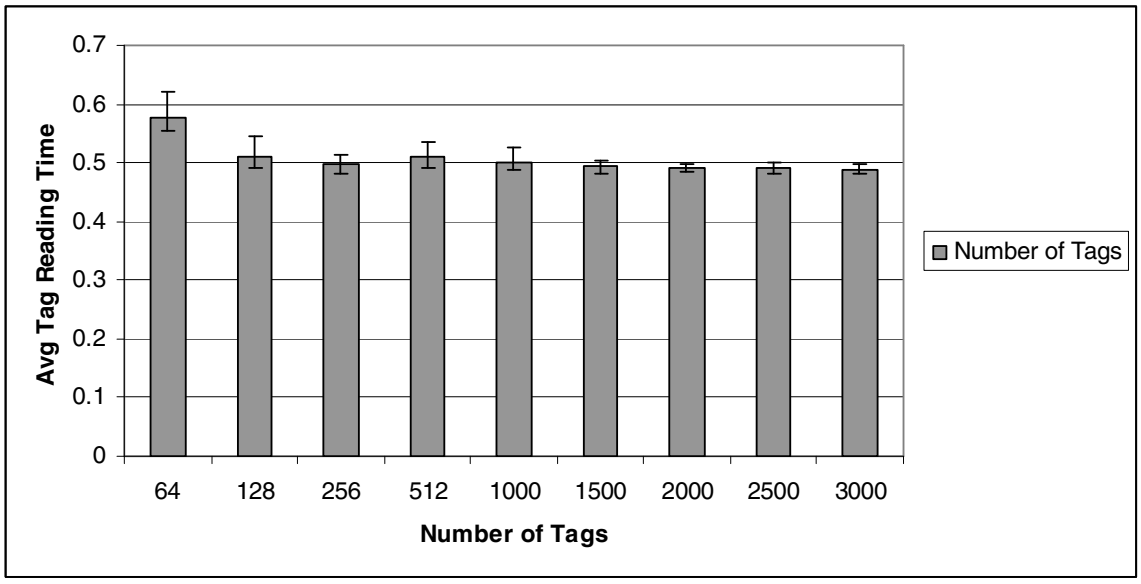

Figure 4.4 Error bar: Average tag reading time vs. n

Figure 4.4 gives the standard deviations in the average tag reading time for various tag count. It is found that the deviations in the results are decreasing as the tag count is increasing which is very useful.

Figure 4.5 gives the statistical standard deviations for the average tag reading time with varying tag count. It includes the data for different values of n. From the figure it is found that the variations in the results are minimal.

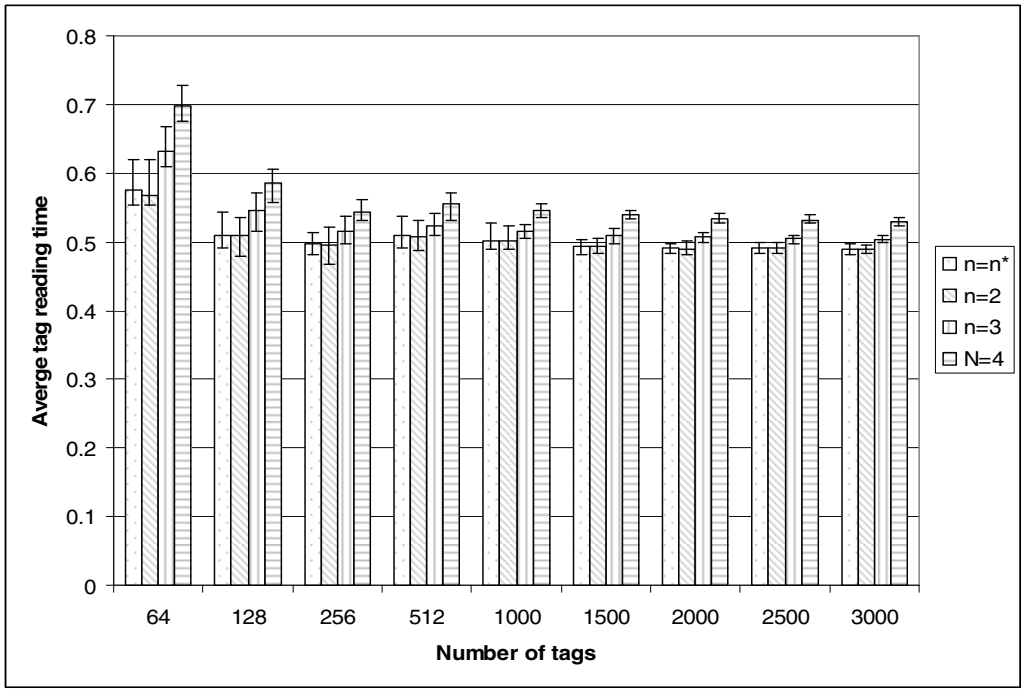

Figure 4.5 Error bar: Average tag reading time for different tag count

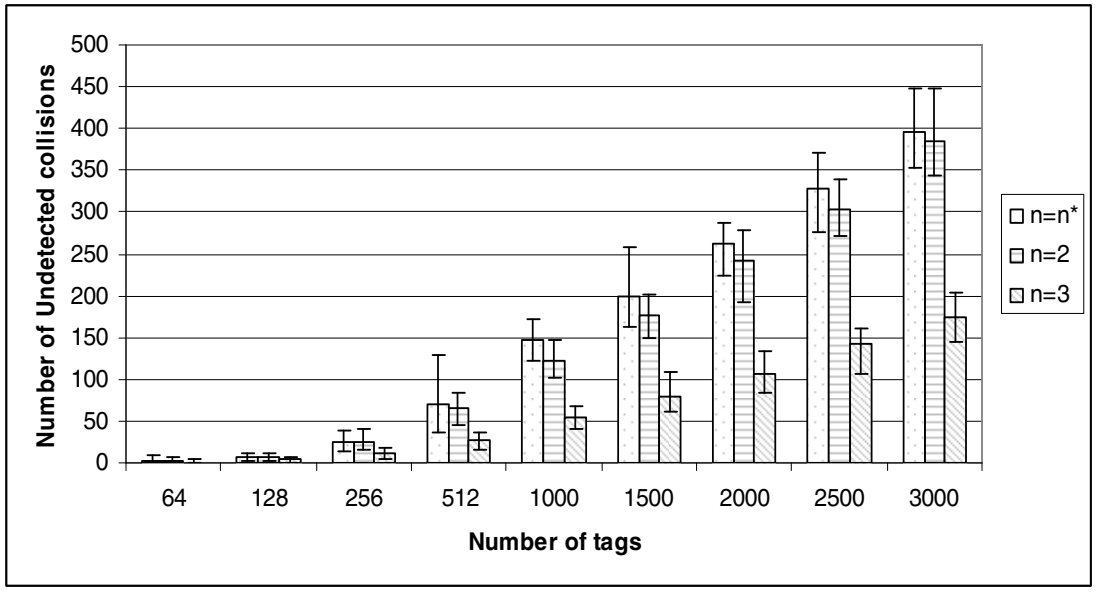

Figure 4.6 Error bar: Number of Undetected collisions with varying n

Figure 4.6 shows the deviations in the number of undetected collisions for n=2, n=3, and n* over a range of tag count.

CHAPTER 5

MOBILE AFSA

RFID has found many applications in many areas like manufacturing industries supermarket checkout, baggage tracking in the airports, vehicle identification near tollbooths, managing traffic flow etc. These industries or applications require that RFID systems should be able to read the tags efficiently when the objects are moving. In these situations, tags enter the reader field and leaves in a shot time. The objective of mobile RFID systems is to read these moving efficiently in a time constrained situation.

5.1 System Model and Assumptions

We use the same system model as that as the m-ASAP [1] and the system model is reproduced below. In this model we use 900 MHz EM field RFID system [1], in which the tags come in to RFID reader region on conveyor belt. We assume that the conveyor belt is moving with a constant velocity, v.

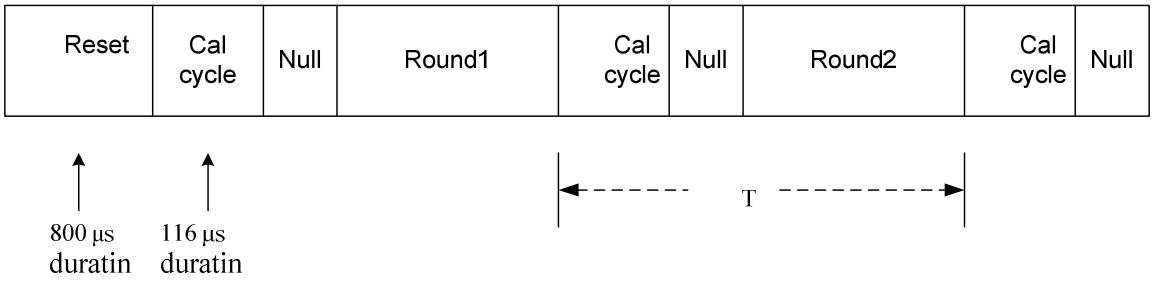

Figure 5-1 m-AFSA round structure

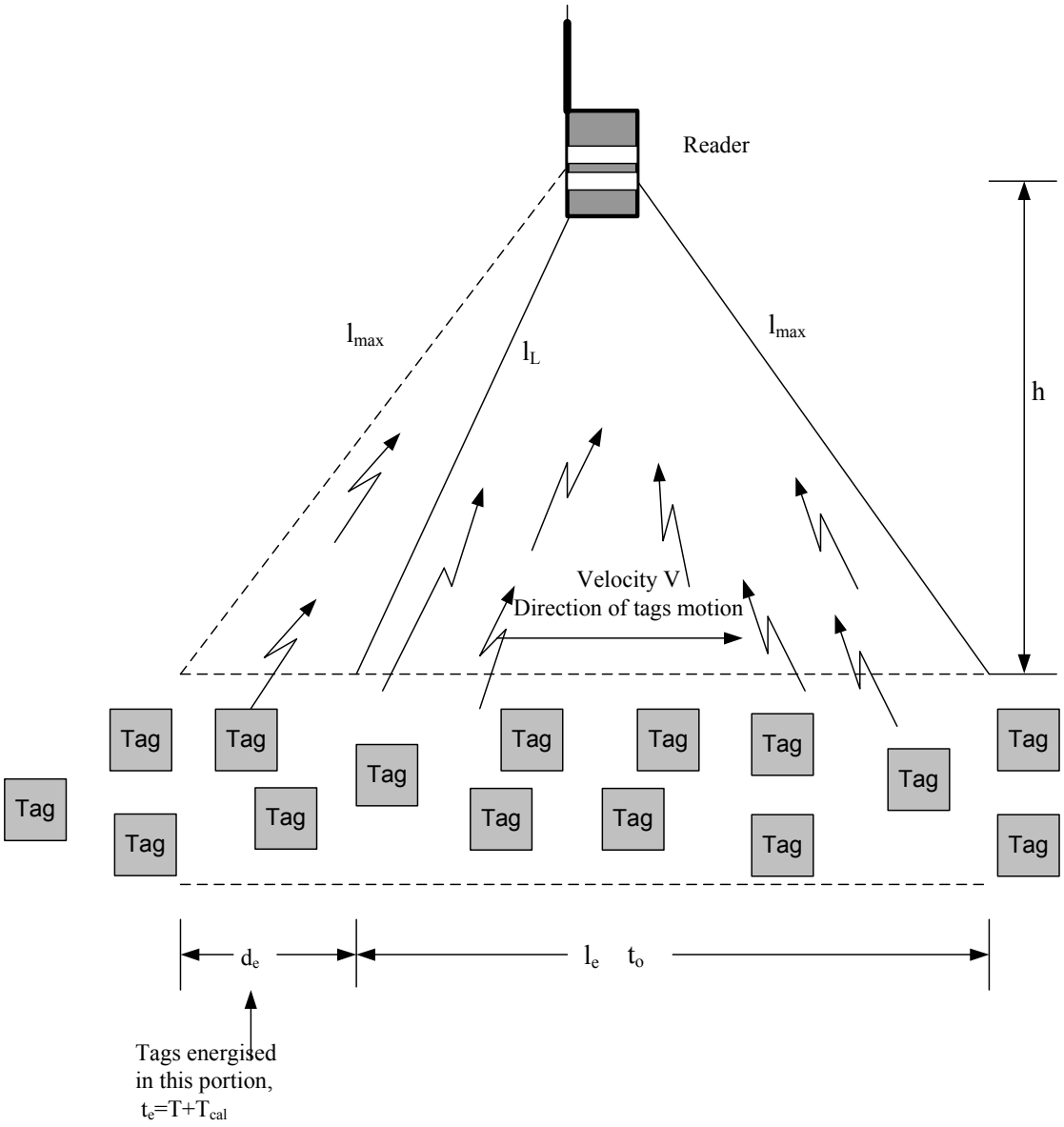

Figure 5-2 Mobile RFID system setup

In the stationary RFID systems, the reader energizes and synchronizes the tags at the beginning of the read process. But in mobile RFID systems, tags may enter the reader field in the middle of read process. Therefore, the system should be such that it can accommodate intermediate power and synchronization [1].

The maximum range of the reader is r_{max} and the vertical distance between the reader and the conveyor belt is if *l*. Tags spend a time of $t = te + to$, in the reader region where, *te* is the time duration in which tags get energized and synchronized with the reader and *to* is the total operating time available for the data transmission before they leave the reader field. Like m-ASAP, we also chose the $te = T + Tcal$, to guarantee that new Bach of tags get at least one calibration cycle.

5.2 Mobile AFSA

In this section, we propose our approach using the above discussed system settings so that it can read desired P% of the total tags.

Mobile AFSA, we use a fixed frame size of 256 and fixed number of rounds as opposed to the m-ASAP. We regulate the tag arrival (ψ) rate accordingly and the initial count to achieve the desired success percentage. We calculate the round by assuming that at most 354 tags [5] tags participate in a round.

$$T = Tcal + Tnull + 8Tb + Tnll + 8Tb + Tnull + 2N*Tb + Tnull + Tnull + N*Tb + Tnull + SB1*320 + Tnll + SB1*Tb + Tnll \quad (5.2\text{-}1)$$

Where Tb is the symbol duration of a bit (12.5 μ sec) and T null is the symbol duration of the null symbol (12.5 μ sec). If we use these values in the equation 5.2-1, we get the T value as 52.3831 m sec.

We know that if we have B1 number of tags to be read and N frame size to offer, estimated number of success slots or tags is given by,

$$E(S1) = B1(1 - \frac{1}{N})^{G-1} \quad (5.2\text{-}2)$$

If we add the same amount new tags to the batch,

$$\psi T = E[S] \tag{5.2-3}$$

From the EDFSA algorithm discussion we knew that it has a maximum success rate of 36.8% in each set given in Table 2-1. If we consider the whole set (using the average success), we can read 89 tags using the frame size 256.

Probability that a tag in the batch B1 (initial count) is not being read after the n rounds is given by

$$1 - p = (1 - \frac{89}{B1})^n \tag{5.2-4}$$

From the equation 5.2-4,

$$B1 = (\frac{89}{1 - (1 - p)^{1/n}}) \tag{5.2-5}$$

Equation 5.2-3 and 5.2-5 gives the tag arrival rate and the initial tag count to achieve a desired read percentage.

5.3 Numerical Results

We ran the simulations for p = 99%, 99.9%, and 99.99%. We stop the simulation once 50000 tags are entered in to the reader region. If we set, conveyor belt speed as 5 m/s, l = 1m and r max = 2m, we will get t = 692.82 m sec.

Number of rounds ((n_r) that the reader needs to use can be calculated using 5.2-1 and the above data.

Table 5.1 Performance results of the m-AFSA

Target read %	Nr (m AFSA)	Nr (BFSA)	B1 (Initial Tag count)	Ψ (tags/sec)	Achieved read % (m AFSA)	Achieved Read % (BFSA)
99	12	7	279.25	1794.1	99.8	98.46
99.9	12	7	203.35	1758.3	99.92	98.8
99.99	12	7	166.09	1661.6	99.99	99.43

Table 5.1 gives the performance results (achieved read percentage) and the required parameter settings for different target read percentages. The achieved tag reading percentages are higher than the target tag read percentages.

CHAPTER 6

CONCLUSION

6.1 Thesis summary and Contributions

In this thesis, we have analyzed the performances of the several Framed Slotted ALOAHA based Medium Access Control protocols and a Binary Search Tree protocol for the RFID systems. We have proposed a framework which can be used for stationary and mobile settings of RFID systems. We have used this framework in conjunction with the Enhanced Dynamic Frame slotted ALOAHA protocol to achieve a better average tag reading time, resulting in a new protocol called Accelerated Frame Slotted ALOAHA. Using simulations, we have proved that AFSA has a better average tag reading time in stationary settings. We also extended our approach to the mobile environment where the tags move in the reader region with a constant velocity. Finally, we have presented the simulation results for mobile RFID systems.

6.2 Future Work

One of our main goals while devising the protocols was to set the optimal value for n so as to minimize the number undetected collisions. From the figure 4.1, we have ≈13% of the total tags colliding during the over all read process. Further refinement of n* value might result in better average tag reading time.

REFERENCES

[1] G. Khandelwal. Efficient design of dense and time constrained rfid systems, Master's thesis, Dept. of Electrical Engineering, Penn State University, August 2005.

[2] D. Gaurav, J. Brain, Panchalingam, Mukunthan, and S. Chris. The use of radio frequency identification as a replacement for traditional bar coding, Tech. Rep., 2004.

[3] H. Vogt. Multiple object identification with passive rfid tags, in *IEEE International Conference on Systems*, vol. 3, pp. 6–9, Man and Cybernetics SMC'02, October 2003.

[4] J. R. Cha, J.H. Kim. Dynamic framed slotted aloha algorithms using fast tag estimation method for rfid systems, *IEEE Communications Society*, pp. 768–772, 2006.

[5] S. R. Lee, S. D. Joo, and C. W. Lee. An enhanced dynamic slotted aloha algorithm for rfid tag identification, *Proceedings of the SecondInternational Conference on mobile and Ubiquitous Systems*, 2005.

[6] Draft protocol specification for a 900 mhz class 0 radio frequency identification tag, Tech. Rep., Auto-ID Center, February 2003.

[7] K. Finkenzeller. RFID Handbook: *Radio-Frequency Identification Fundamentals and Applications*. John Wiley & Sons, 2000.

[8] EPC Global Inc. http://www.epcglobalinc.org/.

[9] S. T. S. Bukkapatanam. *Automation and RadioFrequency Identification in Manufacturing Systems*, *Text book, to be published,* Oklahoma State University.

[10] G. Avoine, P. Oechslin, RFID traceability: A multilayer problem, in *Financial Cryptography – FC'05*, LNCS, Springer, 2005.

[11] S.Jain, and S.R. Das. Collision avoidance in a dense RFID network, *Proceedings of the 1st international Workshop on Wireless Network Testbeds, Experimental Evaluation &Amp; Characterization*, Sept 2006.

GLOSSARY

AFSA: Accelerated Framed Slotted ALOHA

AIDC: Automatic Identification and Data Capture

ASAP: Adaptive Slotted ALOHA.

BFSA: Basic Frame Slotted ALOHA.

Collision slot: Is a slot occupied by only one tag

DFSA: Dynamic Frame Slotted ALOHA.

EFSA: Enhanced dynamic Frame Slotted ALOHA.

EPC: Electronic Product Code.

Frame: A Frame is a group of time slots.

Idle slot: Is a slot which is not being occupied by any tag.

MAC: Medium Access Control.

Reader: RFID reader which tries to retrieve data from RFID tags such as their EPC.

RFID: Radio Frequency Identification.

Successful slot: Is a slot which is occupied by single tag.

Tag: RFID passive tag which has no battery in it and with extremely limited memory and computation capabilities.

Unsuccessful slot: Is a slot which is being occupied more than one tag at the same time.

Vicinity: It is the area covered by RFID reader.

APPENDIX

CODE LISTING

STATIC AVERAGE TAG READING

```
#include<iostream>
#include<stdio.h>
#include<fstream>
#include<iomanip>
#include<math.h>
#include<string.h>
#include<time.h>

#define MAXFRAMESIZE 4096
#define TOTALTAGS 3000
#define TOTALRUNS 25
#define round(x) (x<0?ceil((x)-0.5):floor((x)+0.5))
using namespace std;

int willingness_frame[MAXFRAMESIZE][2]; // to send their willingness
int avail_slot_frame[MAXFRAMESIZE]; // reader uses this frame to tell
which slots are available...
int data_frame[MAXFRAMESIZE]; //actual data reception..but here i am
using only one bit to indiacate transmition...
                                                //here if the count is more
than one means undetected collision in the earlier phase...
int ack_frame[MAXFRAMESIZE];
int groups;
int fsize;
int total_read;
int dead;
int overhead;
int participating_tag_count;
int n_bits;

class tag
{
public:
      int epc; //epc code...
      int random_number; // slot number..
      int willingness; //first level..
      int IntheGroup;//transmittin epc or not
      int sleeping;

      tag();
      void SetTagFields(int);
      void SetGroupNumber();
      int GenerateRandomNumber(int);
```

```
	void SetTheWFrame(int);
	void SetDataFrame(int);


}TagObjects[TOTALTAGS];
tag::tag()
{
	srand((unsigned) time(0));
}
void tag::SetTagFields(int sno)
{
	this->epc = sno;
	this->random_number = -1;
	this->IntheGroup = -1;
	this->sleeping = 0;
	this->willingness = 0;
}
void tag::SetGroupNumber()
{
	int grnd;
	grnd = rand()%groups;
	if(grnd == 0)
		this->IntheGroup = 1;


}
int tag::GenerateRandomNumber(int frame_size)
{
	int rnd;
	rnd = rand()%frame_size;
	return rnd;
}
void tag::SetTheWFrame(int slot_num)
{
	int wrnd;
	int temp=1;
	for(int i=0;i<n_bits;i++)
		temp=temp*2;

	wrnd = rand()%temp;

	if(willingness_frame[slot_num][0]==0) // first attemp in that
slot...
	{
		willingness_frame[slot_num][0]++;
		willingness_frame[slot_num][1] = wrnd;

	}
	else //second attempt in that slot....
	{
		if(willingness_frame[slot_num][1]!= wrnd)
		{
			willingness_frame[slot_num][0]++; // possible
collision...

		}
```

```
	}


}
void tag::SetDataFrame(int slot_num)
{
	if(avail_slot_frame[slot_num]==1)

		data_frame[slot_num]++;
}

class reader
{
public:
	int n_collisions;
	int n_idles;
	int n_success;
	void InitializeReader();
	int EstimateTagCount();
	void SetFrameSizeandGroupNumber(int);
	int SetNumber_of_Bits();
	void SetAvailFrame();
	void SetAckFrame();
	void ReadjustTheCounts();
//	double operator^(double,double);
}r1;
//double reader::operator ^(double x,double y)
//{
//	double z;
//	z = pow(x,y);
//	return z;
//}
void reader::InitializeReader()
{
	this->n_collisions = 0;
	this->n_idles = 0;
	this ->n_success = 0;
}
int reader::EstimateTagCount()//need to work on it...
{
	int ecount = 0;
	double ZI = 0;
	double FS = 0;
	ZI = this->n_idles;
	FS = fsize;
	if(this->n_collisions ==0 && this->n_idles ==0 && this->n_success
==0)
		return ecount;
	else
	{
		if(this->n_idles == fsize)
			ecount=0;
		else
		{
			if(ZI == 0)ZI = 1;
			ecount = (int)(log(ZI/FS)/log((1-1/FS)));
		}
```

```
	}
	if(groups>0)
	ecount = ecount*groups;
return ecount;

}
void reader::SetFrameSizeandGroupNumber(int ecount)
{
	double temp = 0;
	float tt =(float)ecount;

	for(int i=0;i<MAXFRAMESIZE;i++)
	{
		willingness_frame[i][0]=0; // this is for  number of tags
in the slot
		willingness_frame[i][1]=-1;
		avail_slot_frame[i]=0;
		data_frame[i]=0;
		ack_frame[i]=0;
	}

	if(ecount<=5)
	{
		fsize = 4;
		groups = 1;
	}
	else if (ecount>=6 && ecount <=11)
	{
		fsize = 8;
		groups = 1;
	}
	else if (ecount>=12 && ecount <=19)
	{
		fsize = 16;
		groups = 1;
	}
	else if (ecount>=20 && ecount <=40)
	{
		fsize = 32;
		groups = 1;
	}
	else if (ecount>=41 && ecount <=81)
	{
		fsize = 64;
		groups = 1;
	}
	else if (ecount>=82 && ecount <=176)
	{
		fsize = 128;
		groups = 1;
	}
	else if (ecount>=177 && ecount <=354)
	{
		fsize = 256;
		groups = 1;
	}
```

```
	else if (ecount>=355)
	{
		fsize = 256;
		temp = tt/256;
		groups = (int)ceil(temp);
	}



}
int reader::SetNumber_of_Bits()
{
	int n=0;
	double avg = 2;
	double N,k,temp,q2,q3;
	double x;
	double a,b;
	a=2;
	b=64;
	N = (double)fsize;
	k = (double)participating_tag_count;
	temp = (N-1)/N;

	q2 = (k*(k-1)*(1/(N*N))*pow(temp,k-2))/2;
	q3 = (k*(k-1)*(k-2)*(1/(N*N*N))*pow(temp,k-3))/6;




	if(k>2)
	{
		//n = ceil((-N+k*pow(temp,k-
1)+N*pow(temp,k))*(log(N/(b*log(a)*(k-N+N*pow(temp,k)))))/(log(a)*(k-
N+N*pow(temp,k))));


		//n = ceil(log((-b*log(a)*(-N+k*pow(temp,k-
1)+N*pow(temp,k)))/(N))/log(a));


		//n = ceil(log(-b*(avg-1)*(-N+k*pow(temp,k-
1)+N*pow(temp,k)*log(a))/N)/(log(a)*(avg-1)));



		// x = (-8*q2*log(a)+ 2*sqrt( 16*q2*q2*log(a)*log(a) +
2*q3*log(a) ))/(32*q3*log(a));
		 //n = ceil(log(1/x)/(0.6931));


		n = round(log(-17.7446*(-N + k*pow(temp,k-1)+
N*pow(temp,k))/N)/log(a));
		//n = floor(log(-17.7446*(-N + k*pow(temp,k-1)+
N*pow(temp,k))/N)/log(a));
	}
	else
```

```
	{
		n = 2;
	}
	if(n <= 0)
		printf("\n N=%lf,K=%lf",N,k);
 return n;
}
void reader::SetAvailFrame()
{
	for(int i=0;i<fsize;i++)
	{
		if(willingness_frame[i][0]==0) //means idle
		{
			avail_slot_frame[i]=-1;
			this->n_idles++;

		}
		else if(willingness_frame[i][0]>1) //collision
		{
			avail_slot_frame[i]=-1;
			this->n_collisions++;
		}

		else
		{
			avail_slot_frame[i]=1; //success
			this->n_success++;
		}
	}
}
void reader::SetAckFrame()
{
	for(int i=0;i<fsize;i++)
	{
		if(data_frame[i]==1)
		{
			ack_frame[i]= 1; //possitive ack..
		}
		else
			ack_frame[i] = -1; //negative ack...
	}
}
void reader::ReadjustTheCounts()
{
	int temp1 = 0;
	int temp2 = 0;
	int p_acked = 0;
	for(int i=0;i<fsize;i++)
	{
		temp1 = temp1 + data_frame[i];
	}
	for(int i=0;i<fsize;i++)
	{
		if(ack_frame[i]==1)
		{
			p_acked = p_acked +1;
		}
```

```
	}
	temp2 = temp1 - this->n_success;
	this->n_success = this->n_success - temp2;
	this->n_collisions = this->n_collisions + temp2;
	total_read = total_read + p_acked;
	overhead = overhead + temp2;


}


int main()
{
	int estimated_tag_count;
	int Iframe_size=256; // initial frame size...
	int actual_count=TOTALTAGS; // just incase....
	int done;

	int total_slots_used;
	int total_overhead=0;
	float total_time;
	float avg_tag_time;
	float avg_overhead;
	 FILE *fp;
	fp = fopen("output.txt","w");

	n_bits=2;
	srand((unsigned) time(NULL));


	float T_TIME = 0;
	float A_TIME = 0;


	for(int r=0;r<TOTALRUNS;r++)
	{

		groups = 1;
		dead =0;
		overhead = 0;
		participating_tag_count = 0;
		total_read = 0; // number of tags read sucessfully...
		fsize = Iframe_size;

		estimated_tag_count=64;//default for the first round...
		done=0;
		total_slots_used = 0;
		total_time = 0;
		avg_tag_time= 0;




		for(int i=0;i<MAXFRAMESIZE;i++)
		{
			willingness_frame[i][0]=0; // this is for  number of
tags in the slot
```

```
                  willingness_frame[i][1]=-1; // this is to store their
random number...
                  avail_slot_frame[i]= -1;
                  data_frame[i]=0;
                  ack_frame[i]=0;
            }

            r1.InitializeReader(); //reader initialization.....
            for(int i=0;i<TOTALTAGS;i++)  // Initiating the tags...
            {
                  TagObjects[i].IntheGroup = 1;
                  TagObjects[i].random_number = -1;
            }


            //.....................initial round.............//
            total_time = total_time + 113; // for calib
            total_time = total_time + (9 + 8+1)*12.5; // for broad
casting N
            for(int i=0;i<TOTALTAGS;i++)
            {

      TagObjects[i].random_number=TagObjects[i].GenerateRandomNumber(If
rame_size); // generates a random numner to select the slot number..

      TagObjects[i].SetTheWFrame(TagObjects[i].random_number);
//broadcasts n bits to tell the willingness...
            }
            total_time = total_time + (n_bits*Iframe_size)*12.5; // for
willingness....
            r1.SetAvailFrame(); // reader tells which slots are
available...and which are not...
            total_time = total_time + (Iframe_size+8+1)*12.5;  // for
availability
            for(int i=0;i<TOTALTAGS;i++)
            {

      TagObjects[i].SetDataFrame(TagObjects[i].random_number);
            }
            r1.SetAckFrame();
            total_time = total_time + (r1.n_success+ 8 + 1)*12.5;

            r1.ReadjustTheCounts();
            total_slots_used = total_slots_used+fsize - r1.n_collisions
-r1.n_idles;
            total_slots_used = total_slots_used + (4+2)*fsize/64;
            for(int i=0;i<TOTALTAGS;i++)
            {
                  if(ack_frame[TagObjects[i].random_number]==1)
                  {
                        TagObjects[i].sleeping =1;
                        dead++;
                  }
            }

            //........Further rounds.............
```

```
            while(!done)
            {

                  for(int i=0;i<MAXFRAMESIZE;i++)
                  {
                        willingness_frame[i][0]=0; // this is for
number of tags in the slot
                        willingness_frame[i][1]=-1; // this is to store
their random number...
                        avail_slot_frame[i]=-1;
                        data_frame[i]=0;
                        ack_frame[i]=0;
                  }


                  estimated_tag_count = r1.EstimateTagCount(); //reader
estimating the tagcount..
                  //estimated_tag_count = actual_count - dead;
                  if(estimated_tag_count == 0 && r1.n_collisions==0)
                  {
                        done=1;
                        break;
                  }
                  if(estimated_tag_count == 0 && r1.n_collisions!=0)
                        estimated_tag_count = 64;
                  r1.n_collisions =0;
                  r1.n_idles = 0;
                  r1.n_success =0;
                  participating_tag_count = 0;

                  total_time = total_time + 113; // for calib
                  total_time = total_time + (9 + 8 + 1)*12.5; // for
broad casting N

                  for(int i=0;i<TOTALTAGS;i++)  // Initiating the
tags...
                  {
                        TagObjects[i].IntheGroup = -1;
                        TagObjects[i].random_number = -1;
                  }

      r1.SetFrameSizeandGroupNumber(estimated_tag_count);//reader sets
frame size and

                  for(int i=0;i<TOTALTAGS;i++)
                  {
                        if(TagObjects[i].sleeping!=1)
                        {
                              TagObjects[i].SetGroupNumber(); //
calculates the group number which it belongs to...
                        }
                  }
                  for(int i=0;i<TOTALTAGS;i++)
                  {

      if(TagObjects[i].sleeping!=1&&TagObjects[i].IntheGroup==1)
                              participating_tag_count++;
```

```
                    }
                    if(groups == 1)
                    {
                          participating_tag_count = estimated_tag_count;
                    }
                    n_bits=r1.SetNumber_of_Bits();

                    //printf("\t %d",n_bits);
                    for(int i=0;i<TOTALTAGS;i++)
                    {

      if(TagObjects[i].sleeping!=1&&TagObjects[i].IntheGroup==1)// not
already read and in the reading group...
                          {

      TagObjects[i].random_number=TagObjects[i].GenerateRandomNumber(fs
ize);

      TagObjects[i].SetTheWFrame(TagObjects[i].random_number);
                          }
                    }

                    total_time = total_time + (n_bits*fsize)*12.5;// for
willingness....
                    total_slots_used = total_slots_used +
(n_bits*fsize)/64;
                    r1.SetAvailFrame();
                    total_time = total_time + (fsize+1)*12.5;
//availability
                    for(int i=0;i<TOTALTAGS;i++)
                    {

      if(TagObjects[i].sleeping!=1&&TagObjects[i].IntheGroup==1)
                          {

      TagObjects[i].SetDataFrame(TagObjects[i].random_number);
                          }
                    }
                    r1.SetAckFrame();
                    total_time = total_time + (r1.n_success+9)*12.5;
                    r1.ReadjustTheCounts();
                    for(int i=0;i<TOTALTAGS;i++)
                    {
                          if(ack_frame[TagObjects[i].random_number]==1)
                          {
                                TagObjects[i].sleeping =1;
                                dead++;
                          }
                    }
                    total_slots_used =total_slots_used + fsize -
r1.n_collisions -r1.n_idles;


             }

             total_slots_used = total_slots_used + overhead;
             total_time = total_time + (total_read+overhead)*320;
```

```
            total_overhead = total_overhead + overhead;
            avg_tag_time = total_time/(total_read*1000);
            fprintf(fp,"%f\n",avg_tag_time);
            //printf("\n average time for round %d id =
%f",r,avg_tag_time);
            T_TIME = T_TIME + avg_tag_time;

            for(int i=0;i<TOTALTAGS;i++)
            {

                  TagObjects[i].IntheGroup = 1;
                  TagObjects[i].random_number = -1;
                  TagObjects[i].sleeping = -1;

            }


      }
      A_TIME = T_TIME / TOTALRUNS;
      avg_overhead = total_overhead / TOTALRUNS;

      printf("\n average tag reading time is = %f",A_TIME);
      fclose(fp);
      return 0;
}
```

VITA

Malla Reddy Devarapalli

Candidate for the Degree of

Master of Science

Thesis: A FRAMEWORK FOR QUCIK RFID TAG READING IN DENSE ENVIRONMETS

Major Field: Computer Science

Biographical:

Personal Data: Born in Chityala, Andhra Pradesh, India on July 20, 1981

Education:

Received B.S (Engineering) degree in Computer Science from the Osmania University, Hyderabad, Andhra Pradesh, India in 2004. Completed the requirements for the degree of Master of Science with a major in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May, 2007

Experience:

Graduate Assistant (IT Specialist), Food and Agricultural Product Center, Oklahoma State University, Stillwater, Oklahoma

Name: Malla Reddy Devarapalli Date of Degree: May, 2007

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A FRAMEWORK FOR QUICK RFID TAG READING IN DENSE ENVIRONMENTS

Pages in Study: 38 Candidate for the Degree of Master of Science

Major Field: Computer Science

Radio Frequency Identification (RFID) systems provide a mechanism to automatically identify the objects and collect information about them. The main objective of RFID Medium Access Control (MAC) protocols is to provide an opportunity to RFID readers in identifying multiple tags successfully and efficiently. Several variants of ALOHA and Binary Tree Search protocols are being proposed in this area but they show acceptable performances only if limited number of tags are present in the RFID reader region as fairly low amount of data is going to be exchanged. Their performances degrade when large numbers of tags are present in the region because if multiple tags try to communicate with the reader at the same time, it leads to collisions. This problem becomes more complex in the case of mobile tags because of their limited time presence in the reader's region.

In this research, we develop a framework which can be used in conjunction with most of the Framed Slotted ALOHA protocols. A new protocol, Accelerated Framed Slotted ALOHA (AFSA), which is a result of application of the framework with Enhanced Dynamic Frame Slotted ALOHA, not only tries to minimize the number of collisions but also minimizes the total wastage of bandwidth due to collisions and unoccupied slots. We show, through analysis and simulations, our approach gives better average tag reading time over existing models. We further extend our approach to mobile RFID systems where the tags move with a constant velocity in the reader's vicinity.

ADVISER'S APPROVAL: Dr. Venkatesh Sarangan