IMPROVING READER PERFORMANCE OF AN UHF RFID SYSTEM USING FREQUENCY HOPPING TECHNIQUES

By

JU-YEN HUNG

Bachelor of Management in Management & Information

National Open University

Taipei, Taiwan

2006

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, 2009

IMPROVING READER PERFORMANCE OF AN UHF

RFID SYSTEM USING FREQUENCY HOPPING

TECHNIQUES

Thesis Approved:

Dr. Venkatesh Sarangan

Thesis Adviser

Dr. John P. Chandler

Dr. Xiaolin Li

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to take this opportunity to thank all of those who have provided support and direction throughout the creation of this thesis. First, I would like to thank my thesis advisor, Dr. Sarangan, for his patience and guidance. Without his advice and encouragement, the strength of this research would be diminished. Thanks are due to my other committee members Dr. Chandler and Dr. Li for their intuition and encouragement throughout the research process.

To my family and friends, who provided the encouragement I needed through all the trials and the obstacles. I would like to thank my mom and my sister’s family who provide me with a home during vacations. Special thanks to my two daughters for their love and understanding. Special thanks also to my husband; his never ending support has lifted me even when the situation seemed dim.

Finally, thanks to the faculty of the Computer Science Department who pushed me towards hard working. I am grateful that I have made many friends in the department and cherish their friendship.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Introduction

RFID (Radio Frequency Identification), technology used for object tracking and tracing, has been deployed widely over several different fields in recent years. The RFID systems allowing producers and suppliers to scan items in large quantities without line-of-sight, hence saving money and time, have been gradually substituting bar code and commonly recognized as boosting efficiency in supply chain management [1]. However, some of the characteristics of RFID systems, such as large in-flood and inaccuracy, limited the widespread adoption of RFID technology [2].

The data stream generated by RFID readers is not 100% guaranteed; it may suffer from the same problems as most wireless communications-- fading, interference, signal collisions, etc. In the real world, the observed read rate is between 60-70% [2] [3] [4]. An improved performance may have a detection rate of 95-99% [5]. Nonetheless, this read rate is greatly environmentally dependant. The results of readings are usually not as accurate when the processes are done in dense environments. According to the experiment by Wal-mart in 2005, a fully loaded pallet may have its read rate dropped to 66% [6]. As a consequence, collision problems are blamed as the main reasons for deficiency of data reading.

Another factor can be interference. In wireless communications, external interference is not avoidable. The lost information bits due to interference need to be retransmitted later.

Therefore, extra cost is endured. To minimize the cost, the influence of interference must be reduced by using some techniques, e.g. frequency hopping, which is a technique often seen in spread spectrum.

In this paper, we propose a new RFID passive tag reading model using frequency hopping techniques to reduce external interference as well as the number of collisions during the reading process, so that the overall tag reading performance is improved. The anti-collision algorithms will be discussed in chapter 2. Chapter 3 describes how we implement these techniques in our new model. In chapter 4, the simulation results are presented to prove our new model to be both interference and collision resistant. Conclusions will be made in chapter 5.

1.2 RFID basics

A typical RFID system usually consists of some active or passive RFID tags and one or more readers which connect to a backbone computer system [7].

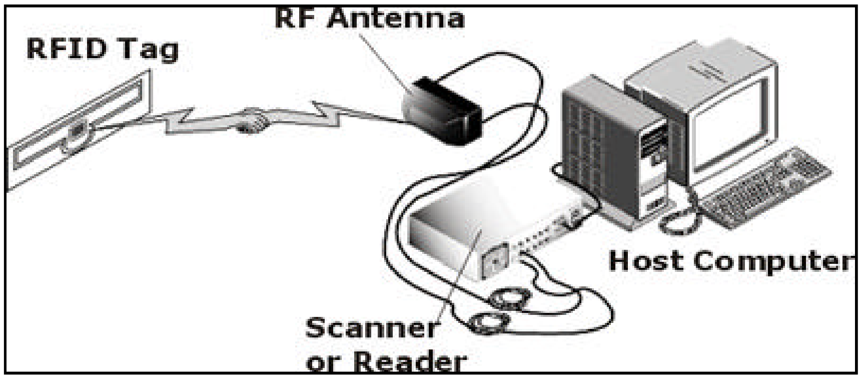

Figure 1. A typical UHF RFID system [7].

RFID tags are small electronic devices consisting of an antenna and a microchip with data capacity of, at most, 2,000 bytes [8]. An active tag contains a battery which can power its microchip; a passive tag has no battery on board and needs an RFID reader providing enough energy to power up the microchip. It is noticeable that the battery in an active tag is reserved for the microchip not for transmitting signals. Whether active or passive, in UHF RFID systems, the tag transmits its information using "backscatter" technology [9]. If all tags backscattered at the same time, the modulated waveforms will be garbled. This is so called tag collision problem [10]. Since the transmission media is air, collision problems greatly influence the reader’s performance. It is important to understand how these problems are encountered. There are three types of collision problems: tag-to-tag, tag-to-reader and reader-to-reader collision problems respectively. Tag-to-reader collision problems can be described as a special case in reader collision problems.

1.2.1 Tag collision problems

When two tags present in the reading zone of a reader, if they send back their information at the same time, the information collides before reaching to the reader. The reader is not able to retrieve either tag’s data. This is called tag-to-tag collision. See figure 2.

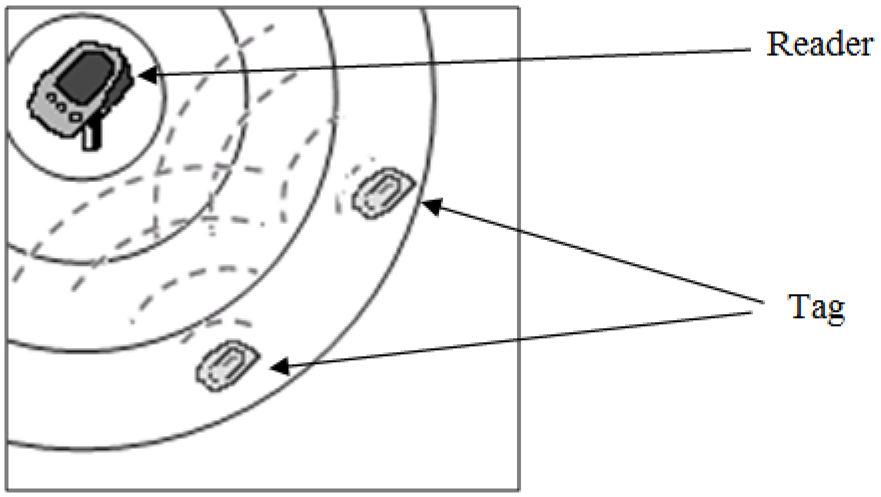

Figure 2. Two tags collides in the same reading zone

1.2.2 Reader collision problems

When a tag is in a reader’s interrogation region, but not far enough from another reader, if the tag is responding to a request from the first reader while the second reader is sending out signals, because the signal strength sent by the second reader is several times stronger than the tag’s signal, the information sent by the tag will be overlapped by the second reader. The first reader either receives incorrect data from the second reader or simply fails to receive any data because of the collision. This is known as the tag-to-reader collision problem, See figure 3.

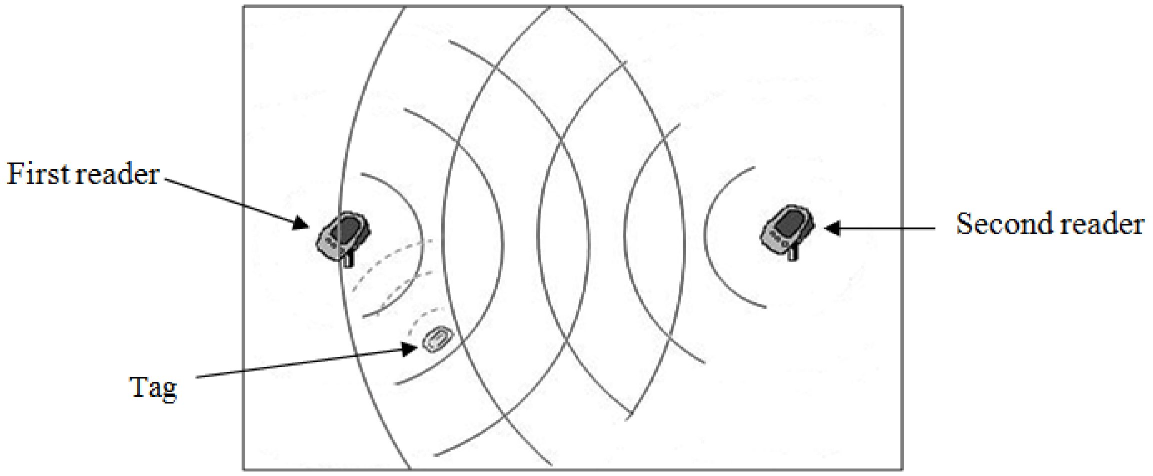

Figure 3 The tag collides with a nearby reader

If the signals sent by the two readers arrive to a tag at the same time, this tag is not able to respond to either reader's request. This overlapped zone is sometimes called a "dead zone". Both readers fail to read this tag. Thus, a reader-to-reader collision occurs. See figure 4.

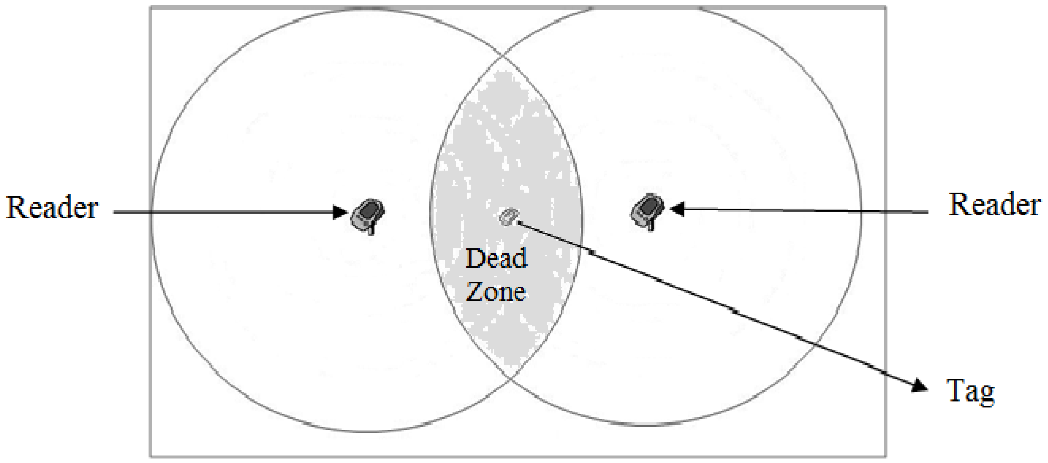

Figure 4 A tag cannot respond to two readers at the same time

Collisions prevent success transmission of information either from reader to tag or tag to reader, thus greatly degrading the efficiency of a system. We will discuss current techniques which are proposed to solve the collision problems in chapter 2.

1.3 Spread Spectrum Communication

1.3.1 Why Spread Spectrum

As mentioned in the beginning of this chapter, the interference in wireless communications is hard to avoid but the impact should be able to reduce using some ways. The idea of spread spectrum technique is to spread the information signals over a wider bandwidth so that jamming and interception of a channel would be more difficult [11] [12]. Depending on how the spectrum is spread and the differences between spread waveforms, the spread spectrum can be identified as frequency hopping spread spectrum, direct sequence spread spectrum, time hopping spread spectrum and chirp spread spectrum. Chirp spread spectrum is used for special purposes and is not as popular as the former three. However, they all share the same benefits of spreading spectrum, which are interference resistance, low probability of intercept (LPI), multipath fading resistance, improved multiple access capability, and ranging [12][13][14].

1.3.2 Frequency Hopping Spread Spectrum

Frequency hopping is a sequence of changing carrier frequencies during signal transmissions. The sequence of hopping from one frequency to the other is called *frequency-hopping pattern*. An example of hopping pattern is showed in Figure 5. The set of possible hopping frequencies is called a *hopset*. Suppose a hopset contains M frequencies, each has a bandwidth of B. The hopping occurs over a hopset called the *hopping band*, whose bandwidth is W, $W \geq MB$. Frequency hopping occurs in time intervals, called *hop interval*. The duration of each hop interval is named *hop duration* or *hop period*. The changing rate of the frequencies is called the *hop rate*.

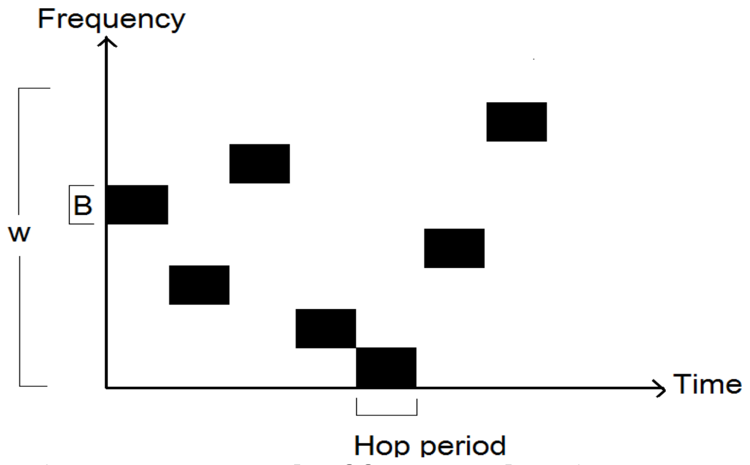

Figure 5. An example of frequency hopping pattern

Frequency Hopping Spread Spectrum, which is the earliest spread spectrum, was the invention of Hollywood star Hedy Lamarr used in military during the World War II. In her invention, the transmitter sends one bit with several frequency hopping intervals. Listening to the channel, whether intentionally or not, will get a sequence of noise like signals. Only the receiver knows which frequency is the priority, after de-spreading, the receiver is able to recover the information sent from the transmitter, hence providing privacy. This is also known as a fast frequency hopping system. See Figure 6.

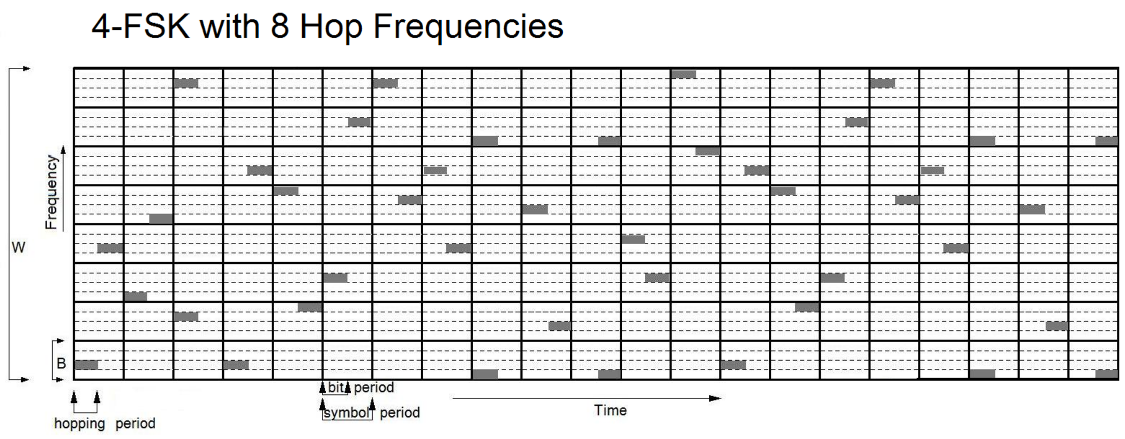

Figure 6. A fast frequency hopping system

Although fast frequency hopping system provides privacy, it is not efficient for transmitting data. That is because there is a switching overhead between each hop. Figure 7 shows the switching overhead.

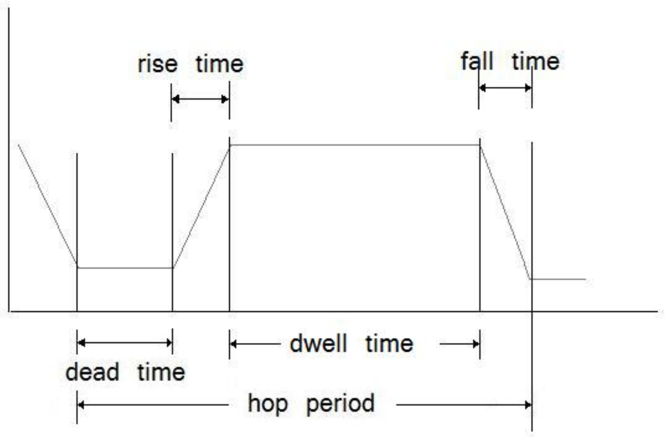

Figure 7. Switching overhead

Switching time is the sum of the fall time, dead time and rise time. The switching time is only used for switching frequencies, during which no information will be transmitted.

Later on, slow frequency hopping system was introduced. A slow frequency hopping system is a system with the hopping period longer than the symbol period, see Figure 8.

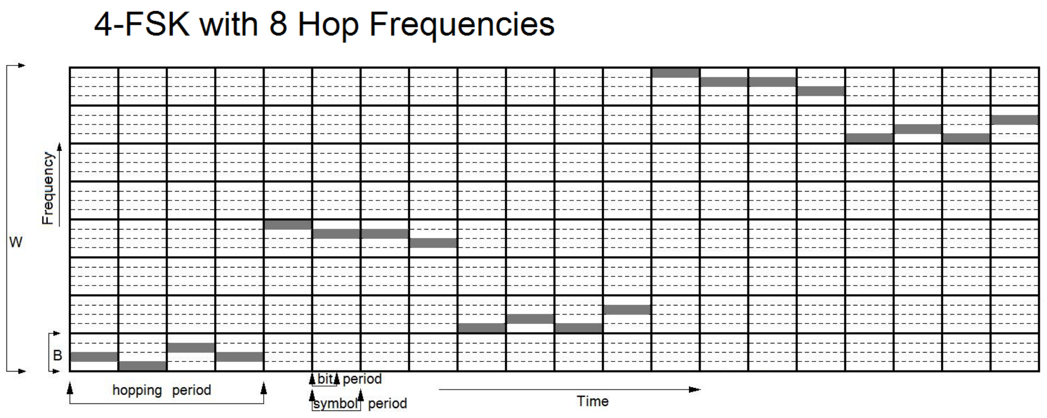

Figure 8. A slow hopping frequency system.

It is obvious that if a piece of information is transmitted as a whole, it is more efficient. But if during the transmission interference is taking place, some information will be garbled. Those lost bits need to be retransmitted to recover the information. A slow frequency hopping system provides interference resistance by nature. During each hopping period a portion of the information will be transmitted. If some channel is jammed or intercepted, the lost information is limited to the portion using that frequency, not the whole piece of information [15] [16]. The faster hopping rate seems to have better interference resistance, but produces more switching overhead, which possibly makes the system less efficient. We will discuss how to determine the ideal hopping intervals so that the overhead and retransmission time are minimal in chapter 3.

CHAPTER II

REVIEW OF LITERATURE

During the querying process of a RFID reader, if multiple tags reply at the same time, it leads to a collision. The limited computation ability of a tag made it hard to communicate among tags to avoid collisions. Instead, the reader takes the responsibility of avoiding collisions. RFID anti-collision protocols can be generally classified as deterministic algorithms and probabilistic algorithms.

Deterministic algorithms, also known as tree based algorithms, prevent collisions by muting most of the tags that are involved. Eventually, there will be a successful transmission from a tag [17]. The reader finished reading all tags in its read zone by visiting them one by one. The advantage of tree algorithms is that the system can obtain higher accuracy, but takes a longer time to read all tags, compared to probabilistic algorithms, especially when a huge number of tags are present at the same time. On the other hand, probabilistic algorithms, including the family of ALOHA based protocols, can read a larger number of tags in a shorter time but in a less accurate manner. There are a lot of extended slotted ALOHA algorithms, some of the most popular will be discussed in the following sections.

2.1 Framed Slotted ALOHA protocols

Framed- Slotted ALOHA (FSA) is the most well known protocol among all deterministic algorithms [18]. By letting each tag transmitting its information to a randomly chosen time slot in a frame, FSA reduces the probability of tag collision. However, if the difference between the frame size and the number of tag counts are large, either idle slots or the number of collisions are also large. This highly degrades the system’s efficiency.

Dynamic FSA (DFSA) [19] and Adaptive Slotted ALOHA Protocol (ASAP) [20] solve this problem by estimating the number of tags present to determine the ideal frame size in the subsequent round. In DFSA, if the tag count is large, the frame size needs to be exponentially increased to identify the tag. Because no matter how many tags remaining unread, it always starts with the initial minimum frame size after identifying a tag [21]. In ASAP, the frame size is determined based on the observation of the previous round. These algorithms work well if the tag counts are small. However, the performance is poor [21] [22] if the number becomes large, because the frame size cannot increase indefinitely as the tag counts increase and the fact that large frame sizes increase the interference between readers in multiple-reader environments. As a result, we need a scheme that can minimize the reading time even if the frame size is limited.

Enhanced DFSA (EDFSA) [23] guarantees a high tag reading rate with a limited frame size by grouping tags to a smaller population so that the probability of a successful reserved slot can be maintained close to 36.8% of the maximum frame size [24]. This approach, however, does not significantly reduce the rounds needed for reading tags.

2.2 Accelerated Framed Slotted ALOHA (AFSA)

The framework of AFSA [24] extends the three phases seen in most slotted ALOHA protocols to five phases. The first phase is the *advertisement phase*, where the reader broadcasts to all tags within its range: the frame size (N), the number of groups (M) and an *n*, which represents the length of an n-bit sequence used for the next phase. A tag first randomly chooses its group number to determine its eligibility to participate in the proceeding round. Each eligible tag then changes its state to “*select*”, and chooses randomly a time slot.

The second phase is the *reservation phase*, during which each tag transmits an n-bit sequence in its chosen slot. There are 2^n possible n-bit sequences, according to the value of n advertised in the previous phase. If an n-bit sequence is received by the reader in a slot, it assumes there is some tag that has successfully reserved that time slot for transmitting its data. If a garbled signal is received, the reader knows there is a collision between two or more tags in that slot.

The third phase is the *reservation summary phase*, in which a bitmap is generated to inform the slot reservation status for tags. A 0 in the i^{th} position of this N-bit summary bitmap indicates either no tag has reserved the i^{th} time slot or a collision occurred in that slot. Nevertheless, a 1 does not guarantee only one tag has chosen that slot. If more than one tag has chosen the same time slot and has transmitted the same n-bit sequence to make the reservation, the reader cannot detect the collision and when those tags transmit data in the later phase, those tags cause a collision. This is called *undetected collision*.

The fourth phase is the *data transmission phase*, wherein all tags that find themselves as successfully reserved statuses transmit their data in the order of the counting of 1s until its position on the bitmap. For example, if the summary bitmap is 0110, the tag that reserved the third time slot should transmit its data second. The rest will go back to "active" and wait for the next advertisement.

The last phase is the *acknowledgment phase*. The reader acknowledges the data transmission from the tags in the form of bits; 0 denotes a failure, 1 denotes a success. A tag receiving a positive acknowledgment will mute itself. Otherwise it goes back to "active" and waits for the next advertisement.

The above five phases are executed sequentially. In order to minimize the average round time, the value of n is limited in the size so that the time for reservation will not be prolonged.

2.3 Advantages and drawbacks for AFSA

AFSA reduces the number of idle slots as well as the number of collisions so that the average tag reading time is reduced by up to 40% with respect to the stand alone ALOHA protocols [24]. It is also found, from the results of simulation, that the optimal value of *n* is 2, which minimizes the total round time when the N and K are known; where K is the participated tag counts for each round. However, by using $n = 2$, we can at most have four different n-bit sequences which produces a large number of undetected collisions that lead to a waste of time slots in the data transmission phase. If we can increase the value of *n* without increasing the total round time, the undetected collision can be reduced and thus improves the performance of the reader.

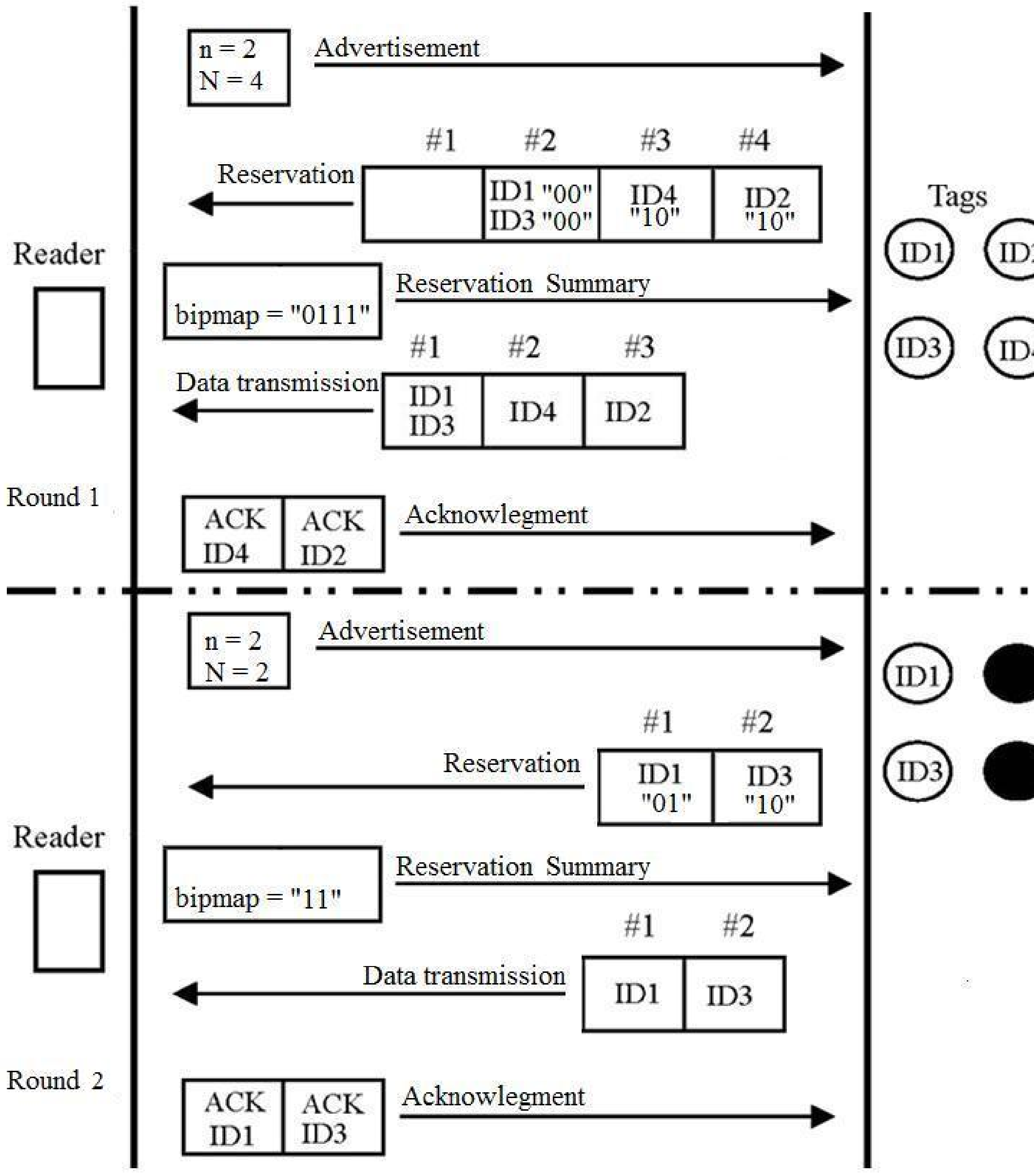

Figure 9. AFSA

CHAPTER III

METHODOLOGY

In our new model, we adapted all assumptions as to AFSA. We are aiming to two goals:

- Reduce the retransmission time caused by external interference.
- Reduce undetected collisions and average tag reading time.

3.1 Reduce retransmission time

We know that if the hopping rate is fast, the bits lost due to interference is less, but the switching overhead increases. On the other hand, if the hopping rate is slow, we lose more information bits due to interference but decrease the switching overhead. How to find a balance point which can minimize the lost bits as well as switching time? Assume a tag contains *b* bits of information, which is divided into *m* portions and modulated to *m* chips during transmission. Each chip period is T_c , where

$$T_c = \delta + b/mR \qquad (1)$$

δ is the switching overhead, R denotes data rate, b/mR is the time that transmits signals (dwell time). If interference occurs at the beginning of transmitting i^{th} chip and continues for T_i seconds, the time for retransmitting the lost bits is kT_c , where

$$k = \text{ceiling}(T_i / T_c) \qquad (2)$$

The total time spent for reading one tag with retransmitting lost bits becomes

$$mT_c+kT_c= T_c\,(m+k)=(\delta + b/mR)\ (m+k) \qquad (3)$$

$$\text{Throughput } S = b/[(\delta + b/mR)\ (m+k)] \qquad (4)$$

Consider some interference occurs with possibility of p, where $0\leq p \leq 1$. The total time spent for reading one tag with retransmitting lost bits is justified as

$$mT_c+kT_c{*}p=T_c(m+kp)=(\delta + b/mR)\ (m+kp) \qquad (5)$$

$$S = b/[(\delta + b/mR)\ (m+kp)] \qquad (6)$$

From above, we found that by using optimal value of $m^*=\sqrt{\frac{128kp}{10}}$, the maximal throughput can be achieved.

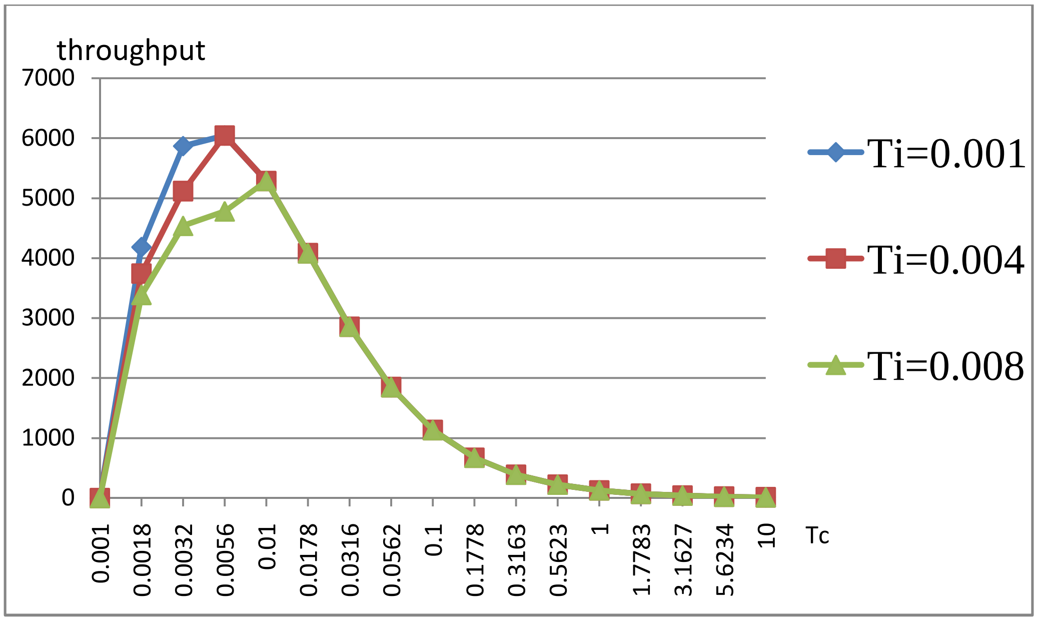

Figure 10. The impact of interference, assuming a probability for interference to occur is 100%, and it continues for T_i seconds.

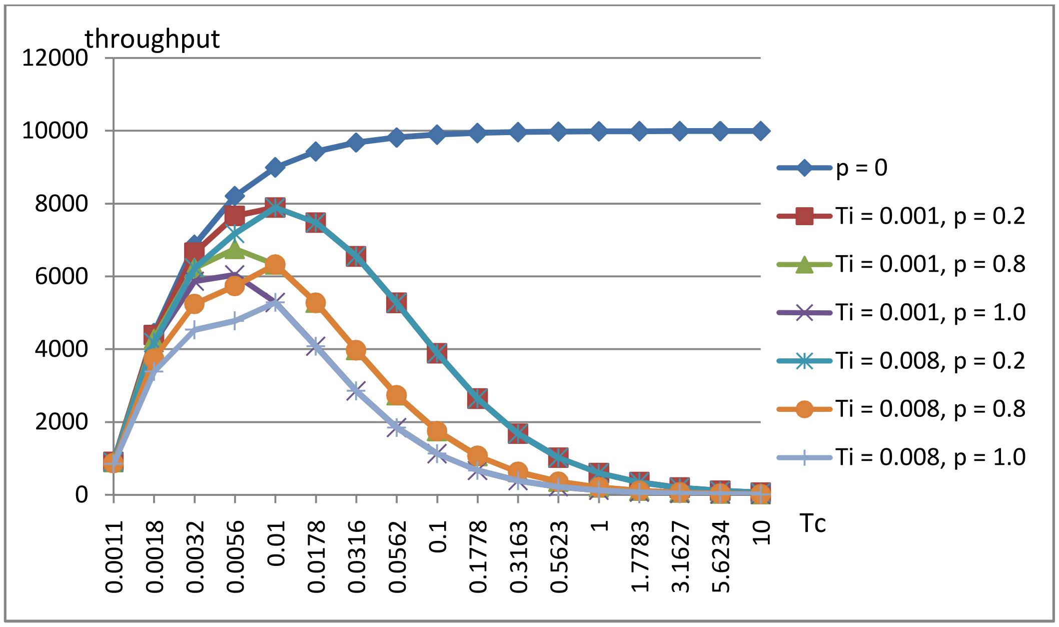

Figure 11. Impact of interference, assuming switching overhead =10 ms, data rate = 10 kb/sec, interference duration T_i seconds with an occurring probability *p*.

Depending on the probability of occurring interference, we found the relationship between the number of portions divided per tag and the duration of interference shown as the following table:

Table 1. Optimized *m* values

Ti	*p*	*k*	*Tc*	*S*	*m*	*m**
0.0001	0.2	4	0.00083	36718.3	4	2.921
0.0001	0.4	6	0.00043	35437.43	8	5.059
0.0001	0.6	6	0.00043	34613.3	8	6.196
0.0001	0.8	6	0.00043	33826.64	8	7.155
0.0005	0.2	4	0.00083	36718.3	4	2.921
0.0005	0.4	4	0.00083	35049.29	4	4.13
0.0005	0.6	4	0.00083	33525.41	4	5.05
0.0005	0.8	4	0.00083	32128.51	4	5.84
0.001	0.2	2	0.00163	35694.37	2	2.065
0.001	0.4	6	0.00043	32355.92	8	5.059

0.001	0.6	6	0.00043	30374.94	8	6.196
0.001	0.8	6	0.00043	28622.54	8	7.155
0.0025	0.2	6	0.00043	32355.92	8	3.577
0.0025	0.4	6	0.00043	28622.54	8	5.059
0.0025	0.6	6	0.00043	25661.59	8	6.196
0.0025	0.8	6	0.00043	23255.81	8	7.155

3.2 Reduce undetected collisions

In the previous study, AFSA executes the 5 phases sequentially. With frequency hopping techniques, we are able to execute these 5 phases in a two-stage pipeline scheme. To implement this model, the reader must be able to monitor both uplink and downlink channels. In other words, the reader should be full duplex, which provides the functionality to transmit and receive data simultaneously. Figure 12 showed an AFSA model without frequency hopping. Figure 13 showed an AFSA model with frequency hopping.

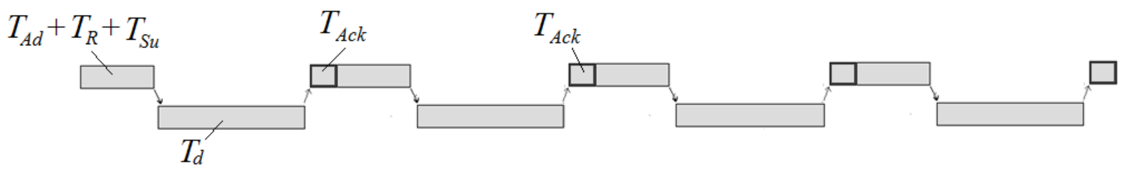

Figure 12. Sequential execution of AFSA.

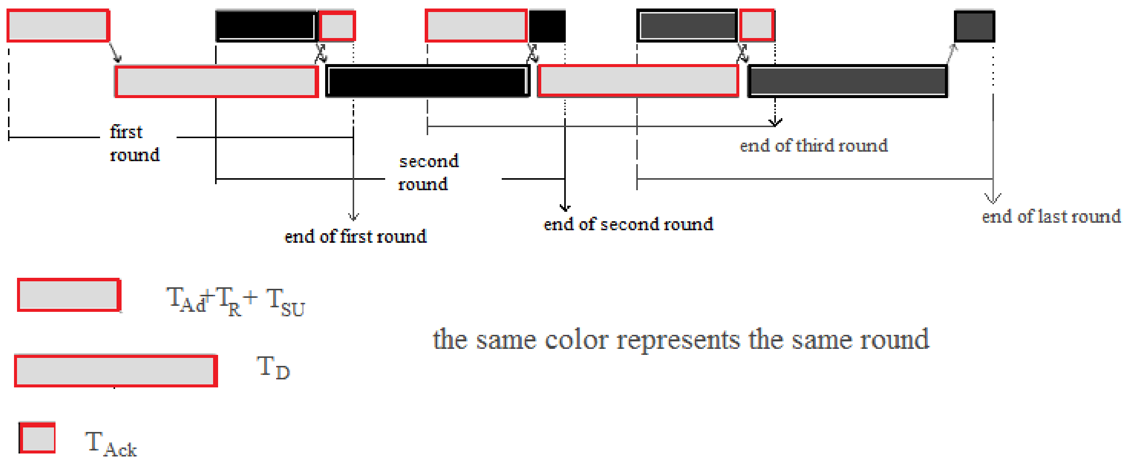

Figure 13. Pipelined execution of AFSA.

Let T_{AD}, T_R, T_{SU}, T_D, and T_{ACK} denote the duration for advertisement, reservation, summary, data transmission, and acknowledgment phases respectively.

$$\begin{aligned} &T_{Ad}=12.5 *(20+ log_2 M+n) \\ &T_R =12.5 * N(n+1) \\ &T_{Su} = 12.5*(10+N) \qquad (7) \\ &T_D = S*(80 * 4+12.5) \\ &T_{Ack} = 12.5*(10+ S) \end{aligned}$$

From [16], we know $S \cong 0.38N$, and $n = 2$ have the best efficiency when executing sequentially. Let T_{SEQ} denote the total time of a round for sequential scheme and T_{HOP} for pipelined scheme. T_{SEQ} can be written as

$$T_{SEQ} = T_{AD}+T_R+T_{SU}+T_D+T_A \qquad (8)$$

Since pipelining will take effect when there is more than one round, we assume the reading takes i rounds. On average, T_{HOP} is

$$\begin{aligned} T_{HOP} &= (T_{AD}+T_R+T_{SU}+T_D+T_{ACK}+(i\text{-}1)*T_D)/i \\ &= T_D+(T_{AD}+T_R+T_{SU}+T_{ACK})/i < T_{SEQ} \qquad (9) \end{aligned}$$

We know that n announced in advertisement phase is the key factor of occurring undetected collision in reservation phase. As n increases, the probability of undetected collisions reduces but durations of advertisement and reservation phases increase. We also noticed that as long as this increasing amount of time is small enough, that is, if

$$T_{AD}+T_R+T_{SU}+T_{ACK} \approx T_D \qquad (10)$$

we can maximize the throughput. From above, the Optimized $n^*=\frac{7.728\mathrm{N}-40}{\mathrm{N}+1}$ can both reduce the number of undetected collisions as well as total read rounds, and further improve the reader performance.

CHAPTER IV

SIMULATION

In this chapter, we present the simulation results that outline the performance of our new model. The source code is listed in the appendix section.

4.1 Specifications

The simulations are done in Java and the results presented in this chapter are the outcomes of 50 different runs. The testing is divided into two portions, first part tests our new model with different *n* values, where *n*=5~8. Each *n* value tests for 50 times with increment of 500 tags and is executed until the unread tag counts less than 2 to provide 99% accuracy. The second part tests and compares AFSA between pipeline scheme and sequential scheme. For each scheme tests for 50 times with increments of 50 tags and are executed until the unread tag counts less than 2 to provide 99% accuracy. In this part, interference is also considered to be possible and the probability of interference is generated randomly by program. For simplicity, a tag will retransmit all its information in case of interference. The results of both portions are outputs of two excel files. Figure 14 showed the diagram of data flow.

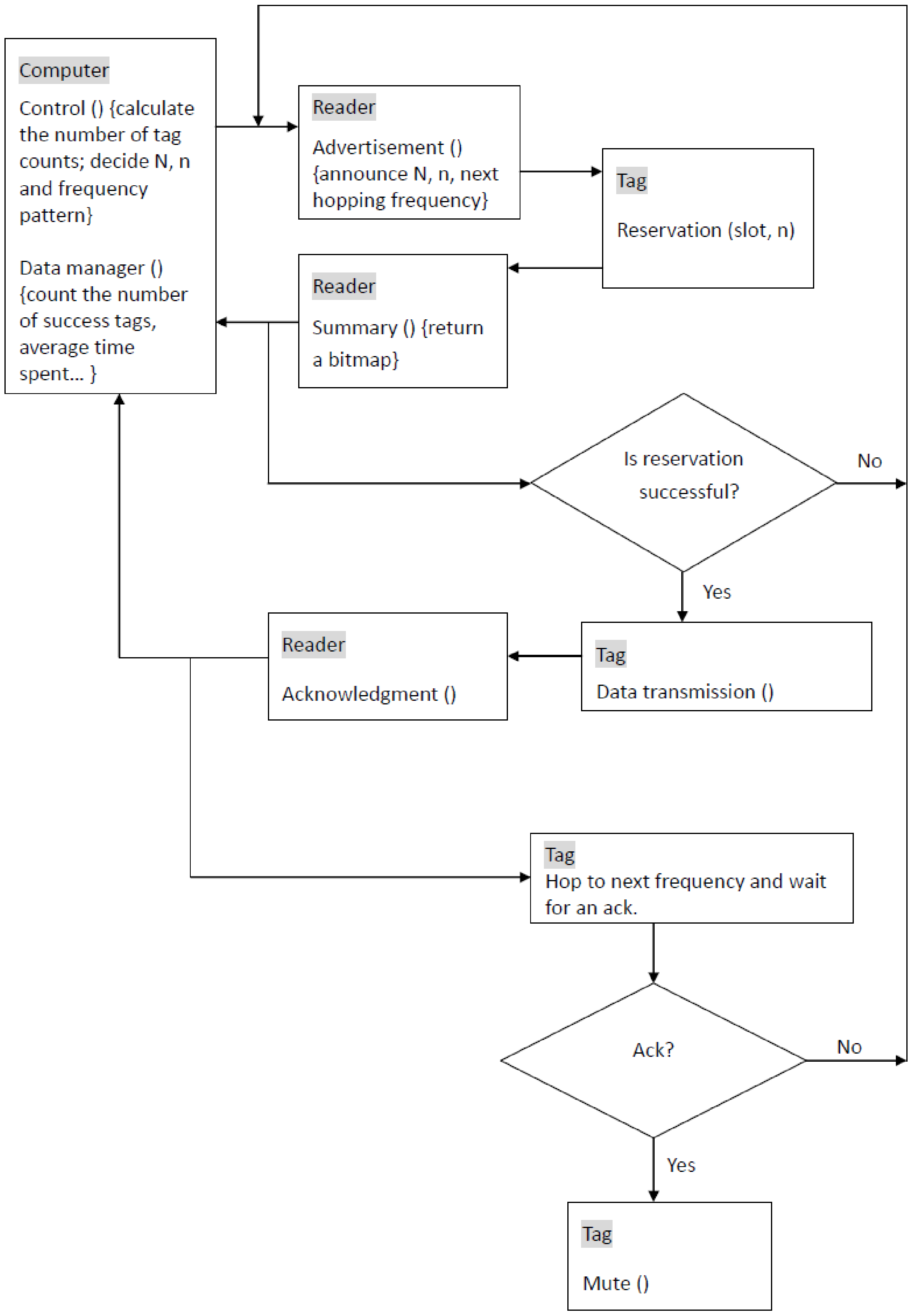
Computer
Control () {calculate the number of tag counts; decide N, n and frequency pattern}
Data manager () {count the number of success tags, average time spent... }
Reader
Advertisement () {announce N, n, next hopping frequency}
Tag
Reservation (slot, n)
Reader
Summary () {return a bitmap}
Is reservation successful?
No
Yes
Reader
Acknowledgment ()
Tag
Data transmission ()
Tag
Hop to next frequency and wait for an ack.
Ack?
No
Yes
Tag
Mute ()

Figure 14. Data flow chart

4.2 Results

As a result of simulations, we have found that using *n**=6 in the pipelined scheme protocol minimized the total reading time for the given number of time slots (Table 2 and Figure 15).

Table 2. Total time spent with different *n* values.

	Total time spent (second)			
tag count	n=5	n=6	n=7	n=8
500	0.189558	0.19144	0.19831	0.2035
1000	0.387063	0.366668	0.366625	0.3785
1500	0.567943	0.53857	0.536013	0.555627
2000	0.76234	0.684205	0.711415	0.76786
2500	0.89745	0.861428	0.88501	0.926433
3000	1.037185	1.033648	1.047045	1.109575
3500	1.23793	1.231615	1.225283	1.330568
4000	1.411162	1.405355	1.43062	1.445057
4500	1.569765	1.52855	1.5463	1.653975
5000	1.73402	1.725595	1.72965	1.852017
:	:	:	:	:
20500	7.109457	6.90744	7.056878	7.581985
21000	7.140802	7.194222	7.2058	7.565645
21500	7.427303	7.308308	7.35098	7.822688
22000	7.74562	7.453395	7.51891	8.019738
22500	7.97172	7.763678	7.656093	8.164128
23000	8.048528	7.905148	7.882303	8.377013
23500	8.11373	7.983488	8.052575	8.59308
24000	8.345117	8.124653	8.288745	8.874668
24500	8.614912	8.27683	8.39088	9.007875
25000	8.603815	8.463832	8.582523	9.084235

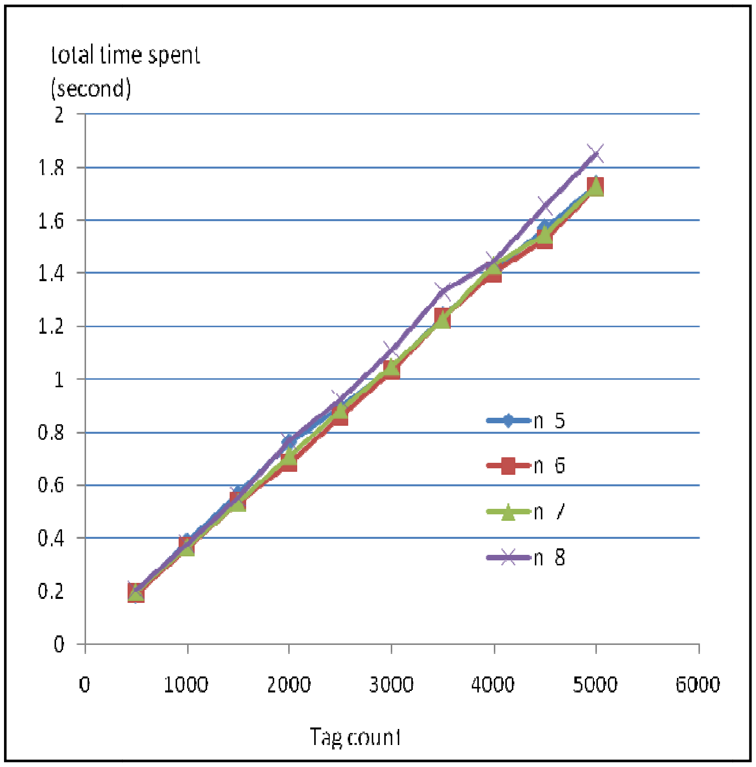

Figure 15. Total time spent with different *n* values.

The tests of pipelined scheme and sequential scheme are using different *n* values. For pipelined scheme, *n*=6, which is based on the results of the first part testing; the sequential scheme uses *n*=2, for it has been proved to be the optimal value for AFSA. We list the results of reading 50~2500 tags using both schemes in Table 3.

Table 3. Average reading time using pipelined and sequential scheme

	Average tag reading time	
Tag counts	Pipelined scheme	Sequential scheme
250	741.18	1769.06
500	725.715	1648.35
750	632.98	1550.753
1000	622.2925	1759.085
1250	724.406	1923.408
1500	776.64	1947.363
1750	669.1257	2119.763
2000	716.0625	2106.715
2250	707.7522	1549.613
2500	639.417	2276.322

2750	683.2455	1633.504
3000	727.115	1550.742
3250	658.8762	1960.914
3500	662.4093	1744.2
3750	734.3507	1710.655

It is obvious that, on average, the pipelined scheme is twice as fast as sequential scheme. Figure 16 showed two very different lines. The pipelined results produce a smoother line, which means it is less influenced by interference; on the other hand, the sequential scheme suffered greatly through interference so that the produced line jumped violently. It proved that the pipelined scheme was more interference resistant and more efficient than the sequential scheme.

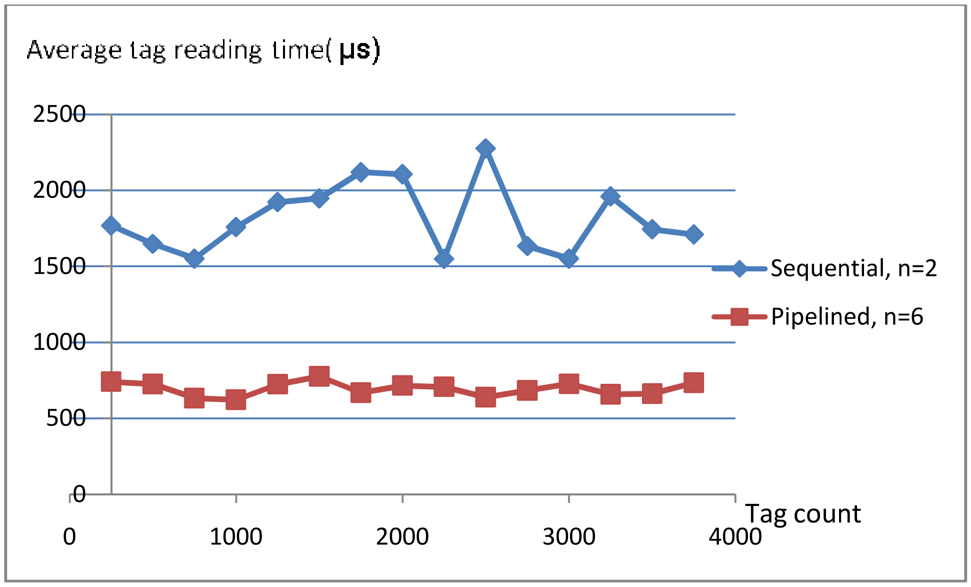

Figure 16. Average reading time using pipelined and sequential scheme

Figure 17 showed the comparison between pipelined and sequential scheme over average tag reading time.

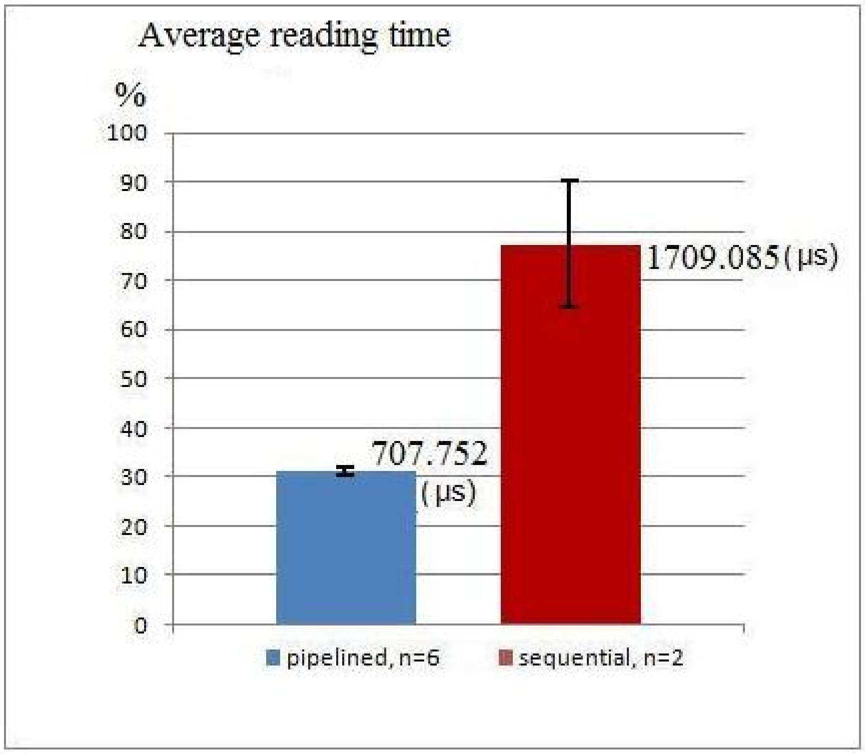

Figure 17. Comparison between pipelined and sequential scheme over average tag reading time

CHAPTER V

CONCLUSION

5.1 Conclusion

The impressive performance of our new model, not only high interference resistance but also high collision avoidance, has proven to increase efficiency by 50 percent on average, compared with sequential execution of AFSA. The key factor is that we execute simultaneously the four phases that are less time consuming with the data transmission phase, which is taking twice as much execution time as the sum of the other four phases. Furthermore, we filled up the time gap between the two pipelined stages with a longer n-bit sequence, which eliminated most undetected collisions.

5.2 Future work

We have proved that with frequency hopping techniques the influence of external interference can be minimized. We also use a two-stage pipeline scheme to cut down the total communication time between reader and tags. In the future, the same scheme can be deploying in mobile environments. It will be a more complex and challenging work, though.

REFERENCES

[1] M. Lee, F. Cheng, and Y. Leung. “A quantitative view on how RFID will improve a supply chain.” *Technical report RC23789 (W0511-065)*, IBM Research Center, November 2005.

[2] R. Derakhshan, M. Orlowska, and X. Li. “RFID Data Management: Challenges and Opportunities.” in *IEEE International Conference on RFID Gaylord Texan Resort*, Grapevine TX, USA, March 26-28, 2007, pages 175-182.

[3] S. Jeffery, M. Garofalakis, and M. Franklin. “Adaptive cleaning for RFID data stream.” *Proceedings of the 32nd International Conference on Vary Large Data Bases (VLDB)*, 2006, pages 163-174

[4] C. Flooremeier and M. Lampe. “Issues with RFID usage in ubiquitous computing applications.” *Pervasive Computing: Second International Conference*, PERVASIVE, 2004. Available at http://*www.vs.inf.ethz.ch/res/papers/RFIDIssues.pdf*

[5] N. Ahmed, R. Kumar, R. S. French and U. Ramachandran, “RFID: A Reliable Middleware Framework for RFID Deployment**.”** in *IEEE International Parallel and Distributed Processing Symposium*, IPDPS March 26-30, 2007, pages 1-10

[6] L. Bolotnyy and D. Evans, “New Directions in Reliability, Security and Privacy in RFID Systems.”Dec. 7, 2007, Available at http://www.cs.virginia.edu/colloquia/event681.html

[7] S. Lewis,”A basic introduction to RFID technology and its use in the supply chain.” *Laran RFID white paper*, January, 2004, page 5. Available at http://www.ship2save.com/page_images/wp_printronix_rfid_supplychain.pdf

[8] K. S. Leong, M. L. Ng, A. R. Grasso, and P. H. Cole, “Synchronization of RFID Readers for Dense RFID Reader Environments,” *Proceedings of the 2006 International Symposium on Applications and the Internet Workshops (SAINT’06)*, 48–51, 2006.

[9] K. Finkenzeller, *RFID Handbook second edition*, Wiley & Sons, 2003, page 29-56, 183-191, 311-314

[10] P. Sorrells, “Passive RFID Basics.” *Microchip Technology Inc*.,1998. Available at http://ww1.microchip.com/downloads/en/AppNotes/00680b.pdf

[11] W. Stallings, “Wireless Communications and Networks” *Prentice Hall,* New Jersey, 2002, page 168-202.

[12] J. Y. Maina, M. H. Mickle, M. R. Lovell and L. A. Schaefer, “Application of CDMA for Anti-Collision and Increased Read Efficiency of Multiple RFID Tags.” in *Journal of Manufacturing Systems,* Vol. 26, 2007, page 37-43.

[13] Y. Fukumizu, M. Nagata, and K Taki, “Back-End Design of a Collision-Resistive RFID System through High-Level Modeling Approach.” In *IEICE Trans, Electron,* Vol. E89-C, No. 11, November 2006, page 1581-1590

[14] R. M. Buehrer, “Spread Spectrum Communications.” *Lecture notes for CEC 5660,* Virginia Tech., 2006, accessible at: http://www.mprg.org/people/buehrer/5660/Lectures/SpreadSpectrumBook.pdf

[15] M. K. Simon, J. K. Omura, R. A. Scholtz and B. K. Levitt, “Spread Spectrum Communications Handbook.” *McGraw-Hill, Inc.,* 2002, page 20-29,

[16] R.M. Buehrer, “Spread Spectrum Communications”, 2007 http://www.mprg.org/people/buehrer/5660/Lectures

[17] W. Chen and G. Lin. “An efficient Anti-Collision Method for Tag Identification in a RFID System” in *IEICE Transactions on Communications,* Vol. E89-B, No. 12 December 2006, 3386-3392

[18] “*EPCTM* Radio-Frequency Identification Protocols Class-1 Generation- 2 UHF RFID Protocol for Communications at 860MHz-960MHz Version 1.0.9,” EPCglobal, Jan. 2005.

[19] J. Cha, J. Kim, “Dynamic Framed Slotted ALOHA Algorithms using Fast Tag Estimation Method for RFID System”, in *Proc. IEEE CCNC 2006,* pp. 768-772.

[20] G. Khandelwal, K. Lee, A. Yener, and S. Serbetli, “ASAP: a MAC Protocol for Dense and Time-Constrained RFID Systems,” *EURASIP J. Wireless Commun. and Networking,* vol. 2007, article ID 18730, 13 pages, 2007. doi:10.1155/2007/18730.

[21] S. Lee, S. Joo, and C. Lee, “An Enhanced Dynamic Framed Slotted ALOHA Algorithm for RFID Tag Identification,” in *Proc. MobiQuitous,* pp. 166-172, July 2005.

[22] H. Vogt, “Multiple Object Identification with Passive RFID Tags.” *2002 IEEE International Conference on Systems, Man and Cybernetics.* October 2002.

[23] H. Vogt, “Efficient Object Identification with Passive RFID Tags.” In *International Conference on Pervasive Computing*, LNCS. Springer-Verlag, 2002. pp.98-113.

[24] V. Sarangan, M.R. Devarapalli and S. Radhakrishnan, “A Framework for Fast RFID Tag Reading in Static and Mobile Environments.” In *Computer Networks journal*, vol. 52, no. 5, April 2008, page 1058-1073.

APPENDICES

Simulation code

1. Main Class

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;

public class Main {

	/**
	 * @param args
	 */

	static int iniCount=0;
	static boolean next=true;
	static PrintStream printStream;
	static PrintStream printStream1;


	public static void main(String[] args) {
		// TODO create an output file, run both sequential and pipelined scheme,
calculate average time
		try {

			printStream = new PrintStream(
					new FileOutputStream(
					new File("result_n.xls")));

			printStream.printf("total time\tn=5\tn=6\tn=7\tn=8\t");
			printStream.println();
			printStream.println("total
count=====================================================");

			printStream1 = new PrintStream(
					new FileOutputStream(
					new File("result_hop.xls")));
```

```
printStream1.println(" average_time pipelined      sequential");
printStream1.println("total_count=========================================
");

                } catch (FileNotFoundException e) {
                        // TODO if the file did not create successfully.
                        e.printStackTrace();
                }

                for(int i=0; i<50; i++){
                        iniCount+=500;
                        runHop(5);
                        Global.timeHop5=Global.totalTimeHop;
                        Global.totalTimeHop=0;
                        runHop(6);
                        Global.timeHop6=Global.totalTimeHop;
                        Global.totalTimeHop=0;
                        runHop(7);
                        Global.timeHop7=Global.totalTimeHop;
                        Global.totalTimeHop=0;
                        runHop(8);
                        Global.timeHop8=Global.totalTimeHop;
                        Global.totalTimeHop=0;
                        Computer.dataManager();
                }

                iniCount=0;
                for(int i=0; i<50; i++){
                        iniCount+=50;
                        runSeq();
                        Global.timeSeq=Global.totalTimeSeq;
                        Global.totalTimeSeq=0;
                        runHop();
                        Global.timeHop=Global.totalTimeHop;
                        Global.totalTimeHop=0;
                        Computer.dataManager1();
                }
        }

        private static void runSeq() {
                // TODO run sequentially with a probability of interference
                Computer.ini();
                Global.unreadTagCount=iniCount;
                double timeSpent=0;

                while(Global.unreadTagCount>3){
                        Reader.ad();
                        Reader.summary(Tag.reservation(Reader.N,2));
                        Tag.transmission();
                        Reader.acknowledgment();
                        timeSpent+=(Global.adTime+Global.suTime+
                                Global.ackTime+Global.reTime+Global.trTime);
                }
                Global.totalTimeSeq=timeSpent;
        }

        private static void runHop() {
                // TODO pipelined scheme with a probability of interference
                Computer.ini();
                Global.unreadTagCount=Main.iniCount;
                Global.n=6;
                Reader.advertisement();//first round
       Reader.summary(Tag.reservation(Reader.N,Reader.nbit));
       Global.totalTimeHop+=(Global.adTimeHop+Global.reTimeHop+Global.suTime);

       //run transmission phase and the previous three phases simultaneously.
       Tag.transmission();//first round
                Reader.advertisement();//second round
                Reader.summary(Tag.reservation(Reader.N,Reader.nbit));//second round
                double
threePhase=Global.adTimeHop+Global.reTimeHop+Global.suTime;

                Global.totalTimeHop+=Math.max(threePhase, Global.trTime);

                //run the rest rounds.
                while(Global.unreadTagCount>3){
                        Reader.acknowledgment();//first round
                        Tag.transmission();//first round
                        Reader.advertisement();//second round
                        Reader.summary(Tag.reservation(Reader.N,Reader.nbit));//second
round
                        double fourPhase=Global.adTimeHop+Global.reTimeHop
                        +Global.suTime+Global.ackTime;
                        Global.totalTimeHop+=Math.max(Global.trTime, fourPhase);

                }

                //last round
                Reader.acknowledgment();//previous round
                Tag.transmission();//this round
                Global.totalTimeHop+=Math.max(Global.trTime, Global.ackTime);
                Reader.acknowledgment();//this round
```

```
                Global.totalTimeHop+=Global.ackTime;
        }

        private static void runHop(int n) {
                // TODO run pipelined scheme
                Computer.ini();
                Global.n=n;
                Reader.ini();
        }
}
```

2. Computer Class

```
public class Computer {

        static void controller(){
                // TODO estimate both frame and n-bit size for the use of advertising

                int numbit=0;
                if(Global.unreadTagCount<256 && Global.unreadTagCount>3){
                        Global.frame=(int)Math.pow(2, log(Global.unreadTagCount));
                        numbit=log(Global.unreadTagCount);
                        if(numbit>Global.n)
                                numbit=Global.n;

                }else if(Global.unreadTagCount<4){
                        Global.frame=2;
                        numbit=1;

                }else{
                        Global.frame=256;
                        numbit=Global.n;

                }
                Global.n=numbit;
        }


        private static int log(int unreadTag) {
                // TODO 2-base log function, this is a supplement function
                int count=0;
                while(unreadTag>1){
                        unreadTag=unreadTag/2;
                        count++;
```

```
		}
		return count;
	}

	static void dataManager(){
		// TODO print out the results with different n-bit length
		String s=Integer.toString(Main.iniCount);

		if(s.length()<4){
			s="  "+s;
		}
		if(s.length()<5){
			s=" "+s;
		}

		Main.printStream.printf("%s\t%3f4\t%3f4\t%3f4\t%3f4\t",
				s,Global.timeHop5*0.000001,Global.timeHop6*0.000001,
				Global.timeHop7*0.000001,Global.timeHop8*0.000001);
		Main.printStream.println();
		Global.timeHop6=0;
		Global.timeHop7=0;
		Global.timeHop8=0;
		Global.timeHop5=0;
	}

	static void ini() {
		// TODO initiate all variables
		Global.unreadTagCount=0;
		Global.frame=0;
		Global.n=0;
		Global.adTimeHop=0;
		Global.adTime=0;
		Global.undetectCollision=0;
		Global.successRes=0;
		Global.suTime=0;
		Global.ackTime=0;
		Global.totalTagRead=0;
		Global.readCount=0;
		Global.reTime=0;
		Global.trTime=0;
		Global.reTimeHop=0;
	}


	static void dataManager1() {
		// TODO print both hopping and sequential scheme results
```

```
            String s=Integer.toString(Main.iniCount);

            if(s.length()<4){
                  s="  "+s;
            }
            if(s.length()<5){
                  s=" "+s;
            }

            Main.printStream1.printf("%s\t%3f4\t%3f4\t",

      s,Global.timeHop/Main.iniCount,Global.timeSeq/Main.iniCount);
            Main.printStream1.println();
            Global.timeHop=0;
            Global.timeSeq=0;
      }
}
```

3. Reader Class

```
import java.util.Arrays;

public class Reader {

      static int N;
      static int nbit;
      static int nextHop;
      static int ack;


      static void advertisement(){
            // TODO advertisement phase
            Computer.controller();
            N=Global.frame;
            nbit=Global.n;
            Global.adTimeHop=12.5*(20+Global.n);
      }


      static int summary(int[][] res){
            // TODO summary phase
            int sucReserTag=0;
            int undetectCollision=0;
            int[] bitmap = new int[N];
            String st="";
```

```
		Arrays.sort(res,new Sort2DArray()); // sort 2D array using comparator to
handle 2'nd dim


		if(N>3){
			for(int i=0;i<N-1 ;i++){
				int l=i+1;

				if(res[i][0]==res[l][0] && res[i][1]==res[l][1] &&
bitmap[res[i][0]]!=-1){
					bitmap[res[i][0]]=1; // undetected collision

					undetectCollision++;
					continue;
				}
				if(res[i][0]==res[l][0] && res[i][1]!=res[l][1]&&
bitmap[res[i][0]]==1){
					undetectCollision--;
					bitmap[res[i][0]]=-1;
				}else if(res[i][0]==res[l][0] && res[i][1]!=res[l][1]){
					bitmap[res[i][0]]=-1;
					continue;
				}
				if(res[i][0]!=res[l][0] && bitmap[res[i][0]]!=-1){
					bitmap[res[i][0]]=1;
				}//if

				if(res[N-2][0]!=res[N-1][0]){
					bitmap[res[N-1][0]]=1;
					if(res[N-2][1]==res[N-1][1]){
						undetectCollision++;
					}
				}
			}//for

		}else if(N<=3 && N>0){
			if(Global.unreadTagCount<2||Main.iniCount<2){
				bitmap[0]=1;
			}else{
			if(res[0][0]!=res[1][0] && res[0][1]!=res[1][1]){
				bitmap[0]=1;
				bitmap[1]=1;
			}else if(res[0][0]==res[1][0] && res[0][1]==res[1][1]){
				bitmap[0]=1;
				bitmap[1]=1;
				undetectCollision+=2;
```

```
                }else{
                        bitmap[0]=0;
                        bitmap[1]=0;
                        }
                }
        }

        for(int j=0; j<N;j++){
                if(bitmap[j]<0)
                        bitmap[j]=0;
                st+=bitmap[j];
                if(bitmap[j]==1){
                        sucReserTag++;
                }
        }
        Global.successRes=sucReserTag;
        Global.undetectCollision=undetectCollision;
        //System.out.println("bitmap = ["+st+"]");
        //System.out.println("undetect collisions = "+undetectCollision);
        //System.out.println("Computer.undetect collisions =
"+Global.undetectCollision);
        Global.suTime=12.5*(10+N);
        res=null;

        return Global.successRes;
    }


    static int acknowledgment(){
        // TODO acknowledgment phase

        if(Global.unreadTagCount>=ack){
                ack=Global.readCount;
                Global.totalTagRead+=ack;
                Global.unreadTagCount-=ack;
        }else{
                Global.totalTagRead=Main.iniCount;
                Global.unreadTagCount=0;
        }
        Global.ackTime=12.5*(10+Global.successRes);
        //System.out.println("Total Tag Read= "+Global.totalTagRead);
        //System.out.println("Total unread Tag= "+Global.unreadTagCount);
        return ack;
    }
```

```
        static void ini() {
                // TODO initiate reader and start the first 2 rounds
                Global.unreadTagCount=Main.iniCount;
                Reader.advertisement();//first round
        Reader.summary(Tag.reservation(N,nbit));
        Global.totalTimeHop+=(Global.adTimeHop+Global.reTimeHop+Global.suTime);

        // pipelined, run transmission phase and the previous three phases simultaneously.
        Tag.transmission();//first round
                Reader.advertisement();//second round
                Reader.summary(Tag.reservation(N,nbit));//second round
                double
threePhase=Global.adTimeHop+Global.reTimeHop+Global.suTime;

                Global.totalTimeHop+=Math.max(threePhase, Global.trTime);

                //run the rest rounds.
                run();
        }


        static void run() {
                // TODO run pipelining scheme
                while(Global.unreadTagCount>3){
                        Reader.acknowledgment();//first round
                        Tag.transmission();//first round
                        Reader.advertisement();//second round
                        Reader.summary(Tag.reservation(N,nbit));//second round
                        double fourPhase=Global.adTimeHop
                        +Global.reTimeHop+Global.suTime+Global.ackTime;
                        Global.totalTimeHop+=Math.max(Global.trTime, fourPhase);
                }
                lastRound();
        }


        private static void lastRound() {
                // TODO last round
                Reader.acknowledgment();//previous round
                Tag.transmission();//this round
                Global.totalTimeHop+=Math.max(Global.trTime, Global.ackTime);
                Reader.acknowledgment();//this round
                Global.totalTimeHop+=Global.ackTime;
        }
```

```
          public static void ad() {
                    // TODO advertisement phase for sequential scheme
                    Computer.controller();
                    N=Global.frame;
                    Global.adTime=12.5*(20+2);
          }
}
```

4. Tag Class

```
public class Tag {

          static int[][] reservation(int slotNum, int num){
                    // TODO reservation phase
                    int[][]reservation =new int[slotNum][2];
                    int slot=0,bit=0;
                    int temp=(int)Math.pow(2, num);

                    if(slotNum>2){
                              for(int j=0;  j<slotNum;j++){

                                        slot=ran(slotNum-1);
                                        bit=ran(temp-1);
                                        reservation[j][0]=slot;
                                        reservation[j][1]=bit;
                              }
                    }else{
                              for(int j=0;  j<slotNum;j++){

                                        slot=ran(slotNum);
                                        bit=ran(temp);
                                        reservation[j][0]=slot;
                                        reservation[j][1]=bit;
                              }
                    }
                    Global.reTimeHop=12.5*(slotNum)*(1+num);
                    Global.reTime=12.5*(slotNum)*3;

                    return reservation;
          }

          private static int ran(int num) {
                    // TODO generate a random integer less than the parameter.
                    int r=0;
                    r=(int)(Math.random()*1000);
```

```
		while(r>num){
			r=Math.abs(r-num);
		}
		return r;
	}

	static void transmission(){
		// TODO transmission phase

		int unsuccessTag=Global.undetectCollision;

		if(Global.successRes>=unsuccessTag){
			Global.readCount=Global.successRes-unsuccessTag;
		}else
			Global.readCount=0;

		Global.trTime=Global.successRes*(12.5+80*4);
	}

	public static void transmission1() {
		// TODO transmission phase with a probability for occurring interference
		double p=Math.random();//Probability for occurring interference
generated by random
		int unsuccessTag=(int)(Global.successRes*p)+Global.undetectCollision;

		//System.out.printf("pro. of interference =  %5.2f",p*100);
		//System.out.print("%\n");
		//System.out.println("# of success Res. Tags = "+Global.successRes);
		//System.out.println("# of unsuccess Tags= "+unsuccessTag);
		if(Global.successRes>=unsuccessTag){
			Global.readCount=Global.successRes-unsuccessTag;
		}else
			Global.readCount=0;

		Global.trTime=Global.successRes*(12.5+80*4);
	}
}
```

5. Global Class

```
public class Global {

	//variable declaration
	static int unreadTagCount;
	static int frame;
```

```
        static int n;
        static double adTimeHop;
        static double adTime;
        static int undetectCollision;
        static int successRes;
        static double suTime;
        static double ackTime;
        static int totalTagRead;
        static int readCount;
        static double reTime;
        static double trTime;
        static double reTimeHop;
        static double timeHop6;
        static double timeHop7;
        static double timeHop8;
        static double totalTimeHop;
        static double timeHop5;
        static double timeHop;
        static double totalTimeSeq;
        static double timeSeq;

}
```

6. Sort2DArray Class

```
import java.util.Comparator;

public class Sort2DArray implements Comparator<Object> {
        public int compare(Object o1, Object o2) {
            int[] a1 = (int[])o1; // second dimension arrays
            int[] a2 = (int[])o2; // must be same length
            for (int i=0; i<a1.length; i++) {     // establish order by comparing
              if (a1[i] < a2[i]) return -1;      // array elements
              else if (a1[i] > a2[i]) return 1;   // from left to right
            }
            return 0; // arrays are equal
        }
}
```

VITA

Ju-Yen Hung

Candidate for the Degree of

Master of Science

Thesis: IMPROVING READER PERFORMANCE OF AN UHF RFID SYSTEM USING FREQUENCY HOPPING TECHNIQUES

Major Field: Computer Science

Biographical:

Personal Data: Born in Taipei, Taiwan.

Education:
Received Bachelor of Management and Information degree in Management and Information from National Open University, Taipei, Taiwan in 2006.
Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May, 2009.

Experience: Teaching Assistant, Department of Computer Science

Name: Ju-Yen Hung Date of Degree: May, 2009

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: IMPROVING READER PERFORMANCE OF AN UHF RFID SYSTEM USING FREQUENCY HOPPING TECHNIQUES

Pages in Study: 40 Candidate for the Degree of Master of Science

Major Field: Computer Science

Abstract:

A new RFID passive tag reading model reducing the average tag reading time in dense environments is introduced. It is shown that by using frequency hopping techniques our model can reduce external interference as well as the number of collisions during the reading processes. The simulation results have further proven that our new model can significantly reduce the average tag reading time by 50%.

ADVISER'S APPROVAL: Dr. Venkatesh Sarangan