Solving the Reader Collision Problem with a Hierarchical *Q*-learning Algorithm

by

Junius K. Ho

S.B. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2001

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

February, 2003

Author:

. .....
Department of Electrical Engineering and Computer Science
February 4, 2003

Certified by:

.....
Sanjay E. Sarma
Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by:

..

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Solving the Reader Collision Problem with a Hierarchical *Q*-learning Algorithm

by

Junius K. Ho

Submitted to the Department of Electrical Engineering and Computer Science on February 4, 2002, in partial fulfillment of the requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis presents HiQ, a hierarchical, online learning algorithm that finds dynamic solutions to the Reader Collision Problem in RFID systems. When the transmissions from one reader interfere with the operation of another reader, a *reader collision* occurs.˚ The objective of the Reader Collision Problem is to minimize the reader collisions experienced by RFID readers while using the minimum number of frequencies and using the minimum total time for all readers to communicate successfully. HiQ attempts to minimize reader collisions by learning the collision patterns of the readers and by effectively assigning˚frequencies over time to ensure neighboring readers do not experience collisions from one another. HiQ is arranged hierarchically with distributed, local control.˚ The algorithm is based on a type of reinforcement learning called *Q*-learning, which is used to determine frequency and time assignments. Through repeated interaction with the system, *Q*-learning attempts to discover an optimum frequency assignment over time.

Thesis Supervisor: Sanjay E. Sarma
Title: Associate Professor

Acknowledgements

First and foremost, I would like to thank my advisor Daniel Engels, without whom this thesis could never have been. The learning process I ve gone through in writing this thesis has been a long and arduous one, with more obstacles than I ever imagined. However, Dan has been there to guide me throughout, with invaluable knowledge and advice. Like my algorithm, I ve had to learn a great deal to accomplish my task; I only wish it could have been as fortunate as I have been to have such a great teacher.

No journey can ever be without a start and a finish, and this thesis began and ended with one man, Sanjay Sarma, who brought me on as a research assistant two Septembers ago, and whose signature on the front of this thesis ultimately validated the fruits of my labor. I am truly grateful for the opportunity he has given me.

To my labmates, Amit Goyal, Noshirwan Petigara and Arundhati Singh, who made the long hours spent in lab both enjoyable and enlightening. I never thought I would actually like the basement of Building 1 but I do, thanks to them.

It s not often that religion, romance and RFID appear in the same conversation, and for that I have Robin Koh and Tom Scharfeld to thank. They ve opened my eyes to the limitless possibilities of the RFID world and provided me with fresh life perspectives.

Thanks to James Waldrop for his help on Colorwave, the elder sibling of HiQ.

A special thanks to my academic advisor, George Verghese for all his support.

Track and field has been a major part of my life at MIT, and I could not have survived without my teammates. However, I owe a great debt to Halston Taylor, the best coach I ve ever had. His lessons off the track were as invaluable as those on the track and I thank him for always believing in me and supporting me.

To the residents of 520 #4, Brandon McKenzie and Aaron Moronez, who somehow made home an entertaining yet restful haven. Every day spent with them was nothing short of hilarious, and I m lucky to live with such great friends.

Hardly anyone ever visited me in lab, with the exception of Lauren Kai. Her laughter and beautiful smile brightened many late nights and her company always lifted my spirits.

Not a day goes by when I don t think about how truly blessed I am to have David and Jean Avrick in my life. I am forever grateful for their undying love and encouragement in all my endeavors. I will never forget the great lengths they have gone to support me.

Last but not least, I would like to thank my parents, Amy and Johnny Ho, and my sister, Alexandra Ho. The lessons they ve taught and the principles they ve instilled are always with me, and I could not be where I am today without them. I am proud to be a part of such a devoted, hard working and loving family.

HiQ

Once they collided

Now they will read without fear

Learning set them free

- Junius K. Ho
February, 2003

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, I present HiQ, a dynamic on-line algorithm for solving the *Reader Collision Problem (RCP)* in Radio Frequency Identification (RFID) systems. First, I motivate the Reader Collision Problem by examining high-density RFID reader applications. Second, I review the fundamentals of RFID systems are reviewed with emphasis on their characteristics that negatively impact their performance in high-density RFID reader applications. Finally, I provide an overview of HiQ, and review its performance.

1.1 Reader Collision Problem and *Q*-learning Overview

The Reader Collision Problem is the fundamental problem of effectively allocating radio frequencies to readers over time such that their interference with one another is minimized. An RFID reader may interfere with the operation of another reader in an RFID system when it communicates. The interference between two operating readers is called a reader collision. Reader collisions prevent the readers from properly communicating with RFID tags. It is important that the number of reader collisions is minimized. Otherwise, the information on the RFID tags cannot be read, and the data is useless.

The Reader Collision Problem looks at minimizing the number of reader collisions in an RFID system though efficient allocation of frequency and time resources. Because of the limited number of resources available in they system, the allocations must be done in an intelligent manner to maximize reader performance.

HiQ utilizes a hierarchical control structure and reinforcement learning [2] based upon *Q*-learning [19] to provide an on-line adaptive solution to the RCP. *Q*-learning tries to solve the Reader Collision Problem through interaction with a network of readers. The *Q*-learning process tries to learn the optimum resource assignment scheme by exploring different assignments and evaluating the performance of these assignments. Through repeated interaction with the environment, *Q*-learning builds a history of the performance of past assignments. Based on this history, *Q*-learning effectively finds a solution to the Reader Collision Problem.

1.2 Introduction to RFID Systems

The two fundamental components in an RFID system are: Radio Frequency (RF) tags and networked RF tag readers. RF tags are typically implemented with low-functionality integrated circuits connected to antennae. The tags store information ranging from simple data types such as static identification numbers, to richer data types like sensory data and product information. The readers are able to communicate with the tags wirelessly and access this data. For example, a shelf full of shaving razors at a supermarket can all be tagged and readers can be placed along the shelves to read the tags. Each of the razors is uniquely identifiable and their status on the shelf (present or

not) can be determined by the readers. Communication between readers and tags is only initiated by readers.

To respond to a reader, a tag needs enough power to communicate with the reader. A tag can power up in two ways. One way a tag can do this is to use its own power supply. Tags with their own, on-tag power supplies are called *active tags.* The second way a tag can power up is to harvest the energy from the signal generated by the reader when it initiates communication with the tag. Tags which use this type of powering up do not have their own, on-tag power supplies and are called *passive tags*. Because of the difference in power levels, active tags have greater communication range than passive tags.

The readers are able to communicate with tags within their respective *interrogation zone*. An interrogation zone is the finite region around the reader where it can communicate with a tag. Due to the nature of radio frequencies, it is almost impossible to create perfectly interlocking interrogation zones. The zones do not form nice squares or rectangles that can be tiled neatly like a bathroom floor, so there is typically zone redundancy within a reader network that covers a large volume. This redundancy creates potential communication problems between the readers, and is the basis of the Reader Collision Problem.

To interact with tags in its interrogation zone, a reader must secure a radio frequency for communication and a time for communication. Typically there are a limited number of distinct frequencies available for use within a reader network. When two readers with overlapping interrogation zones communicate using the same frequency

at the same time, each may experience difficulty in communicating with tags in their interrogation zones due to interfering signals transmitted by the other.

Additionally, readers without overlapping interrogation zones can experience frequency interference. This is because the communication range for a reader signal is much greater than the interrogation zone for a passive tag. Thus, the signal from a tag responding to a reader may be lost due to stronger signals from neighboring readers.

Limited frequency resources require frequency reuse between readers. If these readers are located within the safe reuse distance, it will result in problems from frequency interference. When two readers attempt to communicate using the same frequency, each may interfere with the other s operation. This frequency interference between two readers is one type of *reader collision,* called a *frequency collision*.

A second type of reader collision, called a *tag collision*, may occur due to the communication timings of readers with overlapping interrogation zones. When two readers communicate at the same time, they can experience a tag collision. In a tag collision, two readers attempt to read the same tag at the same time. As a result, both readers may be unable to successfully communicate with the tag. Even if two readers are using different frequencies to communicate with tags in their interrogation zones, there is still a possibility of a tag collision if they are communicating simultaneously.

The consequences of reader collisions are detrimental to the operation of RFID systems. When a reader experiences a frequency collision, it will be unable to communicate with any tags in its interrogation zone. When a reader experiences a tag collision, it may receive erroneous information in communicating with some tags or it may not be able to communicate with the tags at all. The problems resulting from reader

collisions make it imperative that readers minimize the number of collisions they experience.

1.3 Motivation Behind the Reader Collision Problem

RFID systems are used in a variety of applications to uniquely identify physical objects. One of the areas where RFID systems are significantly impacting the overall performance of an application is in supply chain management. At each level of the supply chain — from factories and distribution centers to retail stores — RFID systems can uniquely identify objects and access information about them as they move through the supply chain. From an entire shipment of Coca-Cola to a single razor blade, physical objects can be identified and connected in a Networked Physical World [7]. By leveraging the Internet, shipments can be tracked and inventory can be counted automatically and in real time. Accurate knowledge of object identity and location over time enables supply chain management systems to create an efficient and effective supply chain.

Minimizing the number and frequency of reader collisions is absolutely essential for RFID systems to function properly. In many systems, such as real-time inventory detection and automated product receiving in supply chain management, the area in which tagged objects can be located is quite large. For all of these objects to be read correctly and in a timely manner, readers must be able to detect the tagged objects regardless of their location. To read all tags in a large area such as a warehouse or a supermarket, many readers must be used. This is due to the limits in reader-to-tag communication. Reader-to-tag communication is typically less than ten meters.

Because of these read range limitations, many readers must be placed throughout the area, which creates a highly dense network of readers. When readers are placed in close proximity to other readers, there is the potential for reader collisions. The range of these potential collisions is greater than the range of the interrogation zones of the readers because reader-to-reader interference may be on the order of a few tens of meters. To guarantee that tags in the network are successfully read, the collision rate for readers that are reading must be minimized.

1.4 Defining the Reader Collision Problem

The Reader Collision Problem is the task of assigning radio frequencies to RFID tag readers such that the interference between readers is minimized. The Reader Collision Problem is related to the well-studied Frequency Assignment Problem (FAP) [12] experienced in systems such as cellular telephone systems. Fundamentally, the Reader Collision Problem looks to efficiently assign radio frequency spectrum to a set of radio frequency transmitters (i.e., RFID readers) over time. The optimization metric for the assignment varies depending on the requirements of the application(s) that use the reader collected data. For the Reader Collision Problem, we look to minimize the number of frequencies needed to keep frequency interference at an acceptable level while simultaneously minimizing the amount of time required for all readers to communicate with tags in their respective interrogation zones such that tag interference is kept at an acceptable level. This is very similar to the task of frequency assignment in cellular telephone systems. However, there are more constraints on reader systems due to the minimal functionality of the RF tags.

The Reader Collision Problem can be modeled as a graph coloring problem. The readers are modeled by the vertices of the graph. The edges of each graph represent the interference constraints between two readers. The real world interpretation of two vertices that are connected by an edge is two readers that can interfere with each other if operated at the same time. In other words, two connected vertices can collide. Furthermore, there are two types of edges corresponding to the two types of interference that may be experienced by a reader.

Finding the optimum solution to the Reader Collision Problem is similar to finding optimum solution to a T-coloring problem [4]. In a T-coloring problem, each vertex has an associated set of colors. A vertex can be colored from its set, just as frequencies and time slots can be assigned to readers. The goal of these problems is to find the optimum color assignment — the one using the minimum number of colors — such that no two vertices connected by an edge are assigned the same color.

In the model of the Reader Collision Problem, each color represents a different resource. The number of resources is not restricted to the number of fixed frequencies available in the system. Using a method called Time Division Multiple Access (TDMA), multiplicatively adds to the number of resources available in a reader network. TDMA divides each frequency into multiple time slots.

The use of TDMA is both desirable and sometimes necessary. Even if each reader has been assigned a different frequency than all of its neighbors, tag collisions may still occur. TDMA guarantees that readers which are communicating during different time slots will not experience tag collisions. With limited resources, two

readers which are using the same frequency can read without collisions if they read using different timeslots.

The goal of an RCP algorithm is to limit the number and frequency of both frequency collisions and tag collisions. By minimizing these collisions, a reader network is able to successfully read more data, more often.

1.5 Solving the Reader Collision Problem

An effective RFID system must be designed such that the number and frequency of reader collisions is minimized. To achieve optimum reader performance, an algorithm must be used to allocate the limited frequency resources over time.

There are three general algorithmic approaches to solving the Reader Collision Problem. One approach is centralized, using a central controller to allocate frequencies and time slots within a reader network. The second approach, the distributed control approach, is performed locally without global knowledge. The third approach is a combination, or hybrid, of centralized and distributed where some decisions are made by a centralized controller while the allocation of frequencies over time is performed at a more local level without global knowledge.

This thesis presents the design of a versatile hybrid algorithm for solving the Reader Collision Problem using a type of reinforcement learning called *Q*-learning [19]. The algorithm is designed with a hierarchical network structure and can have either centralized or distributed control, depending on the implementation. The centralized implementation requires more processing power and storage and may not be possible in some applications.

1.6 Thesis Contributions and Overview

This thesis presents the framework, implementation and performance of HiQ, an algorithm based on *Q*-learning that effectively solves the Reader Collision Problem. By using *Q*-learning in a hierarchically distributed manner, the algorithm effectively allocates time and frequency resources to readers without global knowledge of reader constraints. Additionally, I have implemented and simulated the HiQ algorithm and evaluated its performance on reader networks that vary in size, topology and number of available resources. To overcome locally optimal resource assignments, the *Q*-learning process has been augmented with the use of randomization techniques. HiQ is effective in preventing reader collisions, particularly frequency collisions, for reader networks ranging in size from 9 to 49 readers. HiQ also allows for very high transmission rates, enabling readers to successfully communicate greater than 99% in many cases.

In Chapter 2 of this thesis, I review several existing techniques that may be used to solve the Reader Collision Problem. In particular, I look at three learning algorithms used to solve a problem similar to the RCP, called the Frequency Assignment Problem. Additionally, I examine *Colorwave*, a distributed algorithm for solving the RCP [16][17]. The relevant results for each algorithm are also summarized. In Chapter 3, I give an overview of *Q*-learning and how it has been applied to the Frequency Assignment Problem. The second half of this chapter describes the modifications I have made to use *Q*-learning in the RCP. In Chapter 4, I present the HiQ algorithm. I describe the hierarchy and detail the function of each tier in the hierarchy. This includes the distribution of control and how resources are allocated. In Chapter 5, I give my implementation of HiQ. In this chapter, I present the intricacies of each component in the

system, including additional algorithms for fairness and randomization. In Chapter 6, I present the results of a set of six experiments that I performed using the implementation of HiQ. The results of these simulations measures the performance of HiQ in terms of frequency and tag collision rates, and the rate at which readers are granted resources to communicate with. In Chapter 7, I conclude with final thoughts and a discussion of future work.

Chapter 2

Related Work

We review work related to the Reader Collision Problem with special emphasis on the Frequency Assignment Problem and the *Q*-learning algorithm. We examine a *Q*-learning algorithm for the Frequency Assignment Problem as well as other learning algorithms. We finish by reviewing *Colorwave* a distributed algorithm for solving the Reader Collision Problem.

2.1 Frequency Assignment Problem

The Frequency Assignment Problem is the task of optimally assigning limited radio frequencies to communication links in a network. The Frequency Assignment Problem arises in all systems that use radio frequencies for communication [12]. The constraint and optimization criterion of the Frequency Assignment Problem depends on the network and its application.

There are two main types of algorithms that are used to solve Frequency Assignment Problems. The first is fixed frequency, or channel, assignment (FCA). In fixed channel assignment, nodes are assigned a static set of channels that does not change. The second type of algorithm is dynamic channel assignment (DCA). Unlike fixed channel schemes, the set of allocated channels to a node may change over time. Channels are allocated when needed and there are no fixed assignments.

While the Reader Collision Problem is closely related to the Frequency Assignment Problem, there are three key differences between them. First, the mobile devices that communicate in many radio systems are high functionality devices, while the mobile devices in RFID systems (e.g., passive tags) are very low functionality. Second, RFID systems can control both frequency and time assignments. In Frequency Assignment Problems, this is not the case as only frequency is controlled since the base stations are always communicating on their allocated frequencies. Third, constant communication between readers and tags is not necessary. This makes RFID networks more robust to noise; they can tolerate a great deal of noise and still meet their performance criteria. More importantly, it provides a level of flexibility not experienced by systems modeled by the Frequency Assignment Problem.

2.2 Algorithms to Solve the Frequency Assignment Problem

There have been a great number of algorithms developed for solving the Frequency Assignment Problem. The algorithms vary greatly and have many fundamental differences, depending on the type of Frequency Assignment Problem they are trying to solve. Many rely on centralized control for channel assignment, while others boast distributed control where individual transmitters are responsible for communication.

Some examples of dynamic channel assignment algorithms are neural networks, simulated annealing and genetic algorithms [9] [5] [10].

In a neural network, a system of artificial neurons is mathematically created. In this system, the neurons represent a single cellular base station and a single channel that

base station can use. The constraints and optimization criteria of the system are translated into an energy function, which the neural network tries to minimize. Calculating this energy function and combining it with weighted input parameters results in a state for each neuron. The output of these neurons is based on the internal state of each neuron and determines whether or not a channel can be used in a base station. With this algorithm, a network of 25 base stations with 73 channels each was able to converge to an optimum assignment [9].

Simulated annealing is a generalization of local search. The algorithm starts with an initial solution and begins making random changes to channel assignments in a radio network. The algorithm always accepts assignment changes which improve the overall reward, and accepts changes which do not improve the reward (deteriorations) with some probability. The algorithm uses a control variable T to restrict the number of deteriorations and stops when T reaches its terminal value. Simulated annealing algorithms have been shown to be capable of converging to assignment which did not violate frequency constraints 32%-52% of the time, depending on the complexity of the network [5].

Genetic algorithms are a form of blind local search. In a genetic algorithm approach to channel assignment, a single solution is an assignment scheme for all the base stations. The genetic algorithm takes a set of solutions as the population, and begins the evolutionary process. First, members of the population are selected based on their fitness (performance measure). To avoid converging at local minima (when only the fittest are selected) the selection is done in a biased random manner: fitter members are more likely to be chosen than the weaker members. Through crossover, information is

exchanged between members of the selected population, resulting in reproduction which creates new solutions. During reproduction, genes (base stations) may undergo additional random changes at a mutation rate. The process is repeated until a solution is found where no interference constraints are violated. This solution gives an assignment for the set of base stations. The genetic algorithm approach to the Frequency Assignment Problem converged in a network of 25 cell [10].

2.3 *Q*-learning Implementation for the Frequency Assignment Problem

Q-learning is another learning algorithm that has been applied to the Frequency Assignment Problem. This algorithm uses an online reinforcement learning to dynamically allocate frequencies to transmitting cells in a mobile communication network. The *Q*-learning algorithm uses centralized control and global knowledge of potential interference patterns within a mobile communication system to determine frequency assignments.

The *Q*-learning system consists of a learning agent, which interacts with the environment (the system) to learn which assignment schemes perform best. The algorithm measures the performance of each assignment by measuring the interference levels within the system. This data is converted into a performance evaluator — some scalar reward — and passed to the learning agent for. The results of this *Q*-learning process produce *Q-values*, which the agent uses to try and learn an optimum assignment policy. Through repeated interaction with the environment, the learning agent seeks to maximize the rewards for the system, thus finding desirable frequency assignments.

These rewards are based on neighboring cells which may experience frequency interference. Without information about these interference patterns, the *Q*-learning agent cannot make informed decisions for channel allocation.

The *Q*-learning algorithm performed as well or better in terms of blocking probability for different traffic loads than previous fixed channel assignment schemes like neural networks, genetic algorithms and simulated annealing. Against one of the best DCA algorithms, MAXAVAIL [15], the *Q*-learning algorithm performed comparably in terms of blocking probability through significantly reduced computational complexity. The algorithm achieved blocking levels between 0 and 0.2, depending on the traffic patterns and distribution.

2.4 Colorwave

The Colorwave algorithm is representative of algorithms that use a distributed system with no guaranteed method of communicating between neighboring nodes to manage a TDMA reservation anticollision system among a network of RFID tag readers. It is the only previous algorithm designed specifically for the Reader Collision Problem.

In the Colorwave algorithm, the reader network is modeled as an undirected graph. Vertices represent individual RFID readers, and edges represent collision constraints; if two readers connected with an edge transmit at the same time, they will collide. The goal of the algorithm is to color, with a distributed algorithm, a reader network such that each reader node has the smallest possible number of adjacent nodes with the same color. This approach allows easy reservation of timeslots; a color is a periodic reservation for collision-free transmission of data (i.e. reader-tag

communication). A reader with a queued request for transmission transmits only in its color (timeslot). If the transmission collides with another reader, the transmission request is discarded. Furthermore, the reader randomly chooses a new color and reserves this color, causing all of its neighbors to select a new color. This theoretically clears the timeslot for the next time the reader needs to transmit. This switch and reservation action is referred to as a ‘‘kick.’’ Colorwave additionally attempts to optimize the graph to the smallest number of total colors required to achieve a particular percentage of successful transmissions. Since the Colorwave algorithm can cause a node to change its own color, two color variables are maintained at each node: a current color, and the max_colors at that node.

Each RFID tag reader monitors the percentage of successful transmissions. Five inputs to the algorithm determine when a reader changes its local value of max_colors: two safe percentages of successful transmission, one for increasing max_colors and one for decreasing max_colors; two trigger percentages, where a node will alter max_colors only if a neighboring reader is doing so as well; and MinTimeInColor, a stabilization period after a max_colors change during which a node will not alter its max_colors again.

Chapter 3

Q-learning

We examine the details of *Q*-learning, looking at the basic framework and the intricacies of its formulation. We review an example of a *Q*-learning algorithm used for solving the Frequency Assignment Problem. We conclude with the framework and formulation of a modified *Q*-learning to address the Reader Collision Problem.

3.1 Introduction to *Q*-learning

Q-learning is a form of reinforcement learning [2]. In reinforcement learning, situations are mapped to actions that try to maximize a scalar reward (or minimize a cost). Given an environment with defined situations, a learner can find optimum actions for each situation through repeated interaction with the environment. In *Q*-learning, these situations are modeled as states in a discrete-time, stochastic dynamical system. At each state the learner, or *Q*-learning *agent* selects the best course of action from each state based on *Q-values*. *Q*-values are performance measures which are dynamically calculated based on the cost/reward system for the previous actions taken by the agent.

It has been shown that *Q*-learning has been effective for solving dynamic programming problems [3]. The *Q*-learning algorithm we propose for solving the Reader Collision Problem is based on a *Q*-learning algorithm used for solving the Frequency Assignment Problem.

In this chapter the specifics of the *Q*-learning algorithm, as it applies to RFID networks, are presented.

3.2 *Q*-learning FAP Implementation

An algorithm for solving FAP based on *Q*-learning has been shown to perform well against traditional FAP algorithms [8]. The algorithm has centralized control, with one *Q*-learning agent managing the frequency allocation to each cell in the entire network. The algorithm performed better than the MAXAVAIL algorithm, regarded as one of the better dynamic channel assignment algorithms. The *Q*-learning algorithm was less computationally intensive than the MAXAVAIL algorithm.

3.3 *Q*-learning Algorithm

This section details the *Q*-learning algorithm. First, we present the *Q*-learning formulation given for the FAP algorithm. Next, we present the *Q*-learning algorithm for the Reader Collision Problem. This algorithm is modeled after the aforementioned FAP *Q*-learning algorithm, with modifications to fit the differing parameters of the Reader Collision Problem.

3.4 *Q*-learning FAP Formulation

To begin, the environment is modeled as finite-state, discrete-time, stochastic dynamical system. In this environment we let X be the set of possible states, $X = \{x_1, x_2, \ , x_n \ \}$ and A be the set of possible actions $A = \{a_1, a_2, \ , a_m \ \}$.

Here s what happens at each time instant:

1. The agent detects the state $x_t \in X$

2. Based on the state x_t, the agent picks an action $a_t \in A$ to perform

3. The environment transitions to a new state $x_{t+1} = y \in$ X with probability $P_{xy}(a)$, and a return (cost) r_t is generated.

4. That return r_t is passed back to the agent and we return to step 1.

The goal of the *Q*-learning agent is to find an optimum policy $\Pi^*(x) \in A$ for each state x, which minimizes the cumulative measure of the return $r_t = r(x_t, a)$ received over time. Over an infinite time horizon, the total expected discounted return (cost), or *value function* for a state x is given by

$$V^{\Pi}(x) = E\{\sum_{t=0}^{\infty} \gamma^t r(x_t, \pi(x_t)) \mid x_0 = x\} \tag{1}$$

Where E is the expectation operator and the discount factor is 0 † γ † 1. This can be rewritten as:

$$V^{\Pi}(x) = R(x, \pi(x)) + \gamma \sum_{y \in X} P_{xy}(\pi(x)) V^{\pi}(y) \tag{2}$$

where $R(x, \pi(x)) = E\{r(x, \pi(x))\}$ is the mean value of $r(x, \pi(x))$. The optimum policy, π^* satisfies Bellman s optimality criterion,

$$V^*(x) = V^{\pi^*}(x) = \min_{a \in A}[R(x, a) + \gamma \sum_{y \in X} P_{xy}(a) V^*(y)] \tag{3}$$

Q-learning is ideal for a dynamic channel assignment since the optimum policy can be found without knowing $R(x,a)$ and $P_{xy}(a)$.

For a given policy π, let the *Q*-value (state-action value) be

$$Q^{\pi}(x, a) = R(x, a) + \gamma \sum_{y} P_{xy}(a) V^{\pi}(y) \tag{4}$$

This is the expected discounted cost for performing an action a at state x, then following policy π afterwards. To obtain the optimum policy, let

$$Q^*(x,a) = Q^{\pi^*}(x,a) = R(x,a) + \gamma \sum_y P_{xy}(a) V^{\pi^*}(y) \tag{5}$$

which gives us

$$V^*(x) = \min_{a \in A}[Q^*(x,a)] \tag{6}$$

The value function V^* that satisfies Bellman s optimality criterion is found using Q^*. Q^* can be expressed as

$$Q^*(x,a) = R(x,a) + \gamma \sum_y \{P_{xy}(a)[\min_{b \in A} Q^*(y,b)]\} \tag{7}$$

Q-learning recursively tries to find $Q^*(x,a)$ based on the state information and costs. This requires knowledge of states x_t and y_t (at time t and t+1 respectively), a_t (the action taken at time t) and r_t, the cost due to the action taken at time t. Given this information, we have a Q-learning rule (equation 4):

$$Q_{t+1}(x,a) = \begin{cases} Q_t(x,a) + \alpha \Delta Q_t(x,a) & \text{if } x = x_t \text{ and } a = a_t \\ Q_t(x,a) & \text{otherwise} \end{cases} \tag{8}$$

Here α is the learning rate, and

$$\Delta Q_t(x,a) = \{r_t + \gamma \min_b [Q_t(y_t,b)]\} - Q_t(x,a) \tag{9}$$

It has been shown that if the Q-value of each admissible (x, a) pair is visited infinitely often, and if the learning rate is decrease to zero in over time, then $x \rightarrow$, Q_t(x, a)

converges to $Q^*(x, a)$ with a probability of 1. The learning rate used in the experiments is

$$\alpha = \frac{1}{1 + total\ visits\ to\ state\ x} \tag{10}$$

where state x is a combination of node and the number of available resources, as defined previously.

3.5 *Q*-learning Algorithm for the Reader Collision Problem

The *Q*-learning algorithm for the Reader Collision Problem is based on the structures for the FAP equivalent. In this section, we present the intricacies of *Q*-learning algorithm for the RCP.

3.5.1 State Description

In a network of N nodes and R available resources, the state x at time t is given by

$$x_t = (i, A(i))_t \tag{11}$$

where $i \in \{1, 2, \ , N \ \}$ specifies the index and of the node making a resource request and $A(i) \in \{1, 2, \ , R \ \}$ is the number of available resources to node i at time t.

The set of R available resources is different than the set of available frequencies described in the FAP version. In addition to frequencies, another available resource in RFID networks is time slots. Given a set of F frequencies and L time slots, the total number of available resources in the network is

$$R = F * L \quad (12)$$

By using TDMA to manage communication times, the total resources available in the system increase multiplicatively greater with each additional time slot. It should be noted that at least one time slot is required for frequencies to be used. Instead of $A(i) \in \{1, 2, , F\}$, we now have $A(i) \in \{1, 2, , F * L\}$.

To obtain $A(i)$, the number of unique frequency and time slot combinations available to a node i, several other quantities must be defined. The first is the frequency usage status for node q, $q = 1, 2, , R$, defined as an F x L matrix where

$$u_{j,k}(q) = \begin{cases} 1 & \text{if frequency } k \text{ and time slot } j \text{ are in use in node } q \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

and $q = 1, 2, , F$, and $j = 1, 2, , L$, and $k = 1, 2, , F$. Frequency usage is determined by whether or not a frequency has already been allocated to another node, regardless of interference patterns. Unlike the Frequency Assignment Problem, global knowledge of the constraints on individual nodes is not assumed and may not be available.

Additionally, we have a resource availability matrix $r(q) \in \{0,1\}^{LF}$ where

$$r_{j,k}(q) = \begin{cases} 0 & \text{if frequency } k \text{ and time slot } j \text{ available for use in node } q \\ 1 & \text{otherwise} \end{cases} \quad (14)$$

and q, j and k take on the same values as in the usage status matrix. The resource availability matrix is used to accommodate multiple Q-learning agents arranged in a hierarchy. If a resource has not been allocated, it is not available for use.

We can now find the number of available resources $A(i)$. This is given by a set $s_{j,k}(q)$ where

$$s_{j,k}(q) = \begin{cases} 1 & \text{if frequency } k \text{ and time slot } j \text{ can be used in node } q \end{cases} \quad (15)$$

{ 0 otherwise

and

$$s_{j,k}(q) = r_{j,k}(q) \wedge \neg u_{j,k}(q) \quad q = 1, 2, \ldots, R \tag{16}$$

A resource is considered to be available if and only if it has been allocated **and** it is not being used. Given $s_{j,k}(q)$, $A(i)$ is obtained with the summations

$$A(i) = \sum_{j=1}^{L} \sum_{k=1}^{F} s_{j,k}(i) \tag{17}$$

3.5.2 Actions

The agent performs an action in this environment by assigning a frequency k and time slot j from $A(i)$ to node i s request. Here, a is defined as

$$a = j, k \in \{1, 2, , R \} \text{ and } s_{j,k}(i) = 1 \tag{18}$$

3.5.3 Cost

The return $r(x,a)$ evaluates the cost of the action taken. In the RFID network, the cost is not immediately known. It is based on recorded state over a finite time period. This is different from the original algorithm, which had instant costs based on co-channel interference distances and node separation.

The return is based on the number of requests, request rejections, request grants, frequency collisions, tag collisions, frequency interferers and tag interferers. The first five pieces of information are used to measure the performance of the previous assignment. The interferer numbers are used to infer interference patterns in a node. An interferer is a node which can detect the communication of another node. By relaying

this information, the interfering node gives the learning agent more information to learn from.

The cost is calculated by the equation

$$c(x, j, k) = n_1(j,k)c_1 + n_2(j,k)c_2 + n_3(j,k)c_3 + n_4(j,k)c_4 + n_5(j,k)c_5 + n_6(j,k)c_6 \quad (19)$$

In this cost equation, $n_1(j,k)$ represents the rate of successful communication, given by

$$n_1(x, j, k) = \frac{grants - collisions}{grants} \quad (20)$$

Next, $n_2(j,k)$ and $n_3(j,k)$ are the number of tag interferers and the number of frequency interferers per collision, respectively as calculated by

$$n_2(x, j, k) = \frac{tag\ interferers}{collisions} \quad (21)$$

And similarly

$$n_3(x, j, k) = \frac{frequency\ \ interferers}{collisions} \quad (22)$$

The tag and frequency collision rates are $n_4(j,k)$ and $n_5(j,k)$, calculated by dividing over the total number of grants

$$n_4(x, j, k) = \frac{tag\ collisions}{grants} \quad (23)$$

and

$$n_5(x, j, k) = \frac{frequency\ \ collisions}{grants} \quad (24)$$

Finally, the number of rejections per request, is given by $n_6(j,k)$

$$n_6(x, j, k) = \frac{rejections}{requests} \quad (25)$$

Each part of the reward of the cost is multiplied by a constant sub-cost based on their effect on the system. An ordering relation between c_1, c_2, c_3, c_4, c_5, c_6, should be

maintained where $c_1 < c_2 < c_3 < c_4 < c_5 < c_6$. The sub-cost c_1 can and should take on negative values. Additionally, c_1 and c_2 should be a value less than 1, based on the maximum number of interferers. Next, c_3, c_4, c_5, and c_6 should take on values greater than 1. It is important that the sub-costs relating the number of interferers are small enough that they do not dominate the other parts of the equation. This information is designed to help the *Q*-learning agent infer more information about the interference patterns in the system.

The values of each sub-cost used in the simulations was $c_1 = -5$, $c_2 = +0.005$, $c_3 = +0.020$, $c_4 = +1$, $c_5 = +4$ and $c_6 = +5$. The costs reflect the relative performance degradation of each event. Rejections and frequency collisions are the worst, with tag collisions several factors better.

3.5.4 Next State

In the Frequency Assignment Problem, a transition from different states is caused by two stochastic events, frequency assignments and frequency returns. That is, the previously defined state x_t transitions to state x_{t+1} when a frequency is assigned or when it is returned after use. For the Reader Collision Problem, only the action of granting a resource requests causes state transitions. Even though a resource return may change the number of available resources, this information is used only when reassigning resources.

Chapter 4

HiQ Algorithm

In this section, we detail the HiQ algorithm for solving the Reader Collision Problem. The algorithm combines a hierarchical structure with distributed local control for allocating frequency and time resources to a network of readers. This chapter details the components in each tier of the hierarchy and their function in the system.

4.1 Algorithm Goals

The goals of the HiQ algorithm are twofold. First, HiQ tries to allocate resources to maximize the number of readers communicating at one time. The second goal of HiQ is to minimize the number of collisions these readers experience when communicating.

4.2 Distributed Control

Rather than using a purely centralized control structure, the HiQ algorithm uses a hybrid control structure. The task of assigning resources to nodes is distributed among various controllers, each with local knowledge of the nodes it communicates with. This use of local control allows for greater scalability and fault tolerance because there isn t a single point of failure, as in a purely centralized system.

Given a system with a large number of nodes in a network and/or a large number of available resources, the amount of stored information that is required to carry out the Q-learning algorithm may prohibit the use of a single, centralized controller. In large systems with large sets of resources, the data required for Q-learning can grow to unmanageable sizes, even with great computing power. To address this, control in the system is distributed hierarchically.

4.3 Hierarchy

The HiQ algorithm utilizes a hierarchical control structure with three basic tiers. At the lowest tier are RFID readers. The readers communicate when they have been granted a frequency and a time slot to do so. Readers only have knowledge of the frequency and timeslot they have been allocated. Readers are not intelligent — they make requests at some rate, and communicate only when they are provided the resources to do so. They are capable of detecting collisions with other readers by communicating with other readers within its interrogation zone. This information is also known at the tier above the readers, the reader-level server (R-server) tier. The reader tier consists of only a single level, and the readers communicate directly with only one R-server.

R-servers are responsible for preventing collisions between readers that communicate with it. R-servers are able to accomplish this because they have knowledge of the interference patterns for their readers. This local constraint knowledge is given and it allows an individual R-server to assign resources to its readers in such a way that they do not interfere with each other s communication. Each R-server is responsible for at least one reader in a system. All communication by readers in an R-server s

jurisdiction is overseen by that R-server. R-servers have only local knowledge resources and are only able to allocate frequencies and time slots that it has been allocated. Like the reader tier, the R-server tier consists of a single level.

R-servers are allocated frequencies and time slots by the *Q*-learning servers, or *Q*-servers. An R-server can only allocate frequencies and time slots that have been allocated to it by a *Q*-server. *Q*-servers comprise the tier above R-servers. Each R-server communicates with exactly one *Q*-server. *Q*-servers are the highest tier in the fundamental hierarchy, and can be arranged hierarchically themselves. Regardless of how many *Q*-server tiers there are, there is always a single root *Q*-server. The root Q - Server has global knowledge of all frequency and time slot resources, and is able to allocate them all. By interacting with the servers below, each *Q*-server allocates available frequencies and time slots to those servers. Unlike R-servers, *Q*-servers have no knowledge of constraints between individual readers. This information is inferred through interaction with the servers at the tier directly below.

4.4 Reader-level Servers

The sole responsibility of a reader-level server (R-server) is to allocate frequencies and time slot resources to the reader(s) under its jurisdiction. R-servers are designed to ensure that resource assignments which cause reader collisions are not made. R-servers are allocated their resources from the *Q*-server tier directly above them in the hierarchy. In a single *Q*-server structure, R-servers receive resources directly from the root *Q*-servers.

Within a single R-server s domain, an assignment scheme such that frequency constraints are violated will never occur. However, for the purposes of comparison with the FAP algorithms, tag collisions are allowed because FAP problems have a single time slot.

When an R-server has one or more resources allocated to it, it will attempt to grant these resources to the reader(s) below it. The R-server sequentially processes requests from its reader(s) for a frequency and a time slot. When an R-server responds to a request, it checks its pool of allocated resources to see if there is anything available. If there is at least one frequency and timeslot available to the R-server, it will attempt to grant a frequency and a timeslot to the requesting reader. If the assignment can be made without violating any constraints between that reader and the other readers in the R-server s jurisdiction, a frequency and a time slot are granted. If no legal assignment can be made, the request is rejected or *blocked* and the next request is processed.

To maintain fairness within a network of readers, a priority queue is kept up to date by the R-server. This prevents readers that make frequent requests from greedily getting all the grants, and it changes the order in which requests or processed.

4.5 *Q*-learning Servers

Q-learning servers (*Q* -servers) comprise the highest tier of the hierarchy in this algorithm. *Q*-servers allocate resources to the servers directly below them in the hierarchy. These are typically R-servers, but can also be other *Q*-servers.

4.5.1 Resource Allocation

Q-servers make decisions as to which frequency and time resources to allocate based on the *Q*-learning algorithm described in Chapter 3. Each *Q*-server maintains a *Q*-table, updating entries in the table as the *Q*-server interacts with the servers below. The size of the table is static; it is dependent on the number of nodes the *Q*-server controls, the total number of frequencies and the total number of time slots.

Q-servers allocate resources based on the node that requests the resource and the total number of resources the *Q*-server has available to allocate. If the *Q*-server determines that there is a resource to be allocated, the node which makes the request is allocated the optimum resource as determined by the *Q*-learning mechanism with some probability. To avoid local optima which may be caused by purely deterministic decisions, a *Q*-server may allocate a random resource to the node, rather than always assign the optimum resource.

After a *Q*-server finishes responding to the requests made by the nodes it controls, it waits for each node to respond with its communication patterns using the allocated resource. This is the information that the *Q*-server uses to calculate costs for the previous resource assignment, and it includes the number of successful reads, the number of tag and frequency collisions detected and the total number of frequency and tag interferers detected. The calculated costs are fed back into the *Q*-learning mechanism, and the allocation process is repeated.

4.5.2 *Q*-server Hierarchy

In the previous section, we mentioned that it is possible that a *Q*-server may control another *Q*-server. This is possible because the *Q*-server tier can be arranged hierarchically. The purpose of this hierarchy is to reduce the complexity of control for the *Q*-servers. In a system with a large number of nodes and/or a large number of resources, the sheer size of a *Q*-table can prohibit the algorithm from achieving satisfactory performance. By reducing the number of servers that each *Q*-server is responsible for, the control complexity is significantly reduced, as well as memory requirements.

Given a system consisting of one thousand R-servers, the benefits of using multiply-tiered *Q*-servers over a single *Q*-server tier can easily be seen. With one, singly-rooted *Q*-server tier, one *Q*-server is responsible for all one thousand R-servers. If the total number of resources is large, the *Q*-table for this *Q*-server will significantly degrade its performance. However, if another tier of *Q*-servers is added between the root *Q*-server and the R-servers, the memory requirements and control responsibilities for each *Q*-server are significantly reduced. With just ten additional *Q*-servers responsible for one hundred R-servers each, the root *Q*-server s *Q*-table is reduced by two orders of magnitude, while the size of the largest *Q*-table (found in each server in the second tier of *Q*-servers) is reduced by one order of magnitude.

Chapter 5

HiQ Algorithm Implementation

This chapter outlines the implementation of the three major components —*Q*-learning servers (*Q*-servers), reader-level servers (R-servers) and readers — in the HiQ algorithm. The functions of each component are described in detail, along with additional algorithms that are used to augment their performance.

5.1 Implementation Overview

In a network of *Q*-servers, R-servers and readers, there will be exactly one tier of both the R-servers and the readers. The R-servers are directly responsible for assigning resources to the readers while the *Q*-servers allocate resources to the R-servers to distribute amongst the readers.

In addition to the functions performed by each component, algorithmic enhancements at the *Q*-server and R-server tiers are also described. First, a fair queuing algorithm is implemented in all *Q*-servers and R-servers to prevent components in the tier below them. Second, randomization techniques are used in the *Q*-server to avoid locally optimum solutions.

5.2 Resources: Frequencies and Time Slots

In the implementation of HiQ, both time slots and frequencies are used as resources in the system. However, they are coupled resources in that both a frequency and a time slot must be assigned for a reader to communicate. To ensure that the system behaves correctly, this fundamental requirement must be met.

The total number of time slots in the system is fixed. Information about when readers are allowed to communicate is propagated from the highest *Q*-server tier down to the readers. Regardless of which controller a reader communicates with, the readers communicate using the same set of time slots. If two completely detached readers that received to two different resource grants allocating the same time slot (e.g., time slot 3) communicate during that time slot, their communication starts and ends at the same time. This allows the *Q*-servers to effectively distribute resources.

5.3 Readers

5.3.1 Overview

The readers are the fundamental component in the network. They are simple devices whose main purpose is to communicate with tags in their interrogation zones. The have the least amount of intelligence in the network and are not designed to resolve collision issues. In this algorithm, readers which may experience collisions — those with overlapping interrogation zones (read regions) — have the ability to communicate with each other. This allows readers to figure out whether they are colliding with neighboring readers. The number of collisions a reader may experience is stored by the reader.

To perform a read, a reader needs a frequency and a time slot. These are granted to the readers from the next tier up in the hierarchy. When a reader is granted a frequency and a time slot, it is allowed to search for tags in their interrogation zone, using their allocated frequency during that time slot.

5.3.2 Reader Collisions

One of a reader s main responsibilities is to detect collisions. This includes both frequency and tag collisions. In order to do so, readers will communicate with other readers with overlapping interrogation zones when there is a read occurring. Let s say Readers B and C are within the interrogation zone of Reader A (Figure 5-1). When A is granted a frequency and a time slot, it begins reading. During this time, it also pings its neighbors (B and C) to see if they are reading. Once Readers B and C receive the ping, they respond with whether or not they are reading at the time, and if they are, what frequency they are using.

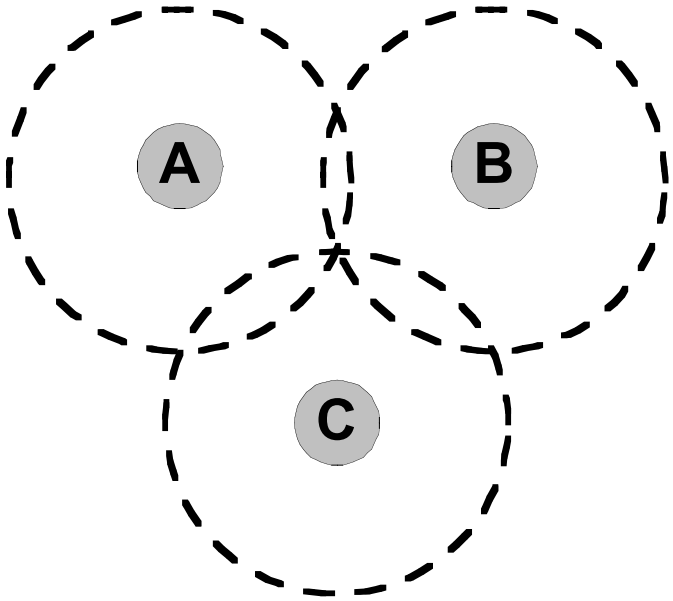

Figure 5-1. Three reader configuration.

The original reader (Reader A in this case) is responsible for the collisions detection processing. When Reader A receives the ping responses from Readers B and C, it checks to see if there are collisions, and if so, what type of collisions. If two readers are communicating using the same time slot, there will be a tag collision. If these readers are also using the same frequency, the reader will also experience a frequency collision. A tag collision can occur without a frequency collision while frequency collisions always indicate tag collisions as well.

Sometimes, a communicating reader will collide with multiple readers. The reader counts this as a single collision so that collisions are not counted multiple times. If all the readers in the three-reader network are reading at the same time using the same frequency, there should be a total of six collisions — three frequency collisions and three tag collisions. If each reader were to count all the collisions it experiences, there would be twelve collisions, twice as many as the correct number. In reality, we only want to know whether or not a reader experienced a tag collision and/or a frequency collision. Each reader can experience at most one of each.

However, the information about how many other readers a communicating reader collides with is useful information. Readers that have a high number of interferers per collision are more generally constrained. This information is stored as the number of tag and frequency *interferers* and is used in the *Q*-learning algorithm as outlined in Chapter 3.

When the reader is done communicating, it sends a message back to the reader-level server with information about any collisions that may have occurred during its read.

If there are no collisions, then the reader does not send a message back to its reader-level server. This saves communication and processing bandwidth because there are fewer messages to be handled.

While the readers are able to detect collisions, they do not have the intelligence to avoid collisions. The tiers above the reader in the hierarchy are responsible for collision avoidance. One of the design goals is to keep the readers as simple as possible, reducing the computational and communication bandwidth used by readers.

5.3.3 Read Rates

There are two possible states that a reader can be in. A reader is either reading, or it is not reading. The rate at which a reader makes requests can vary, creating different traffic patterns. A reader which makes requests 100% of the time means that when it is pinged by an R-server, it will always respond with a request for a frequency and time slot. When readers are always making requests, the Reader Collision Problem most closely resembles the Frequency Assignment Problem, where the nodes communicate every chance they get.

Alternatively, readers may requests a frequency and a time slot with some probability p. This is more indicative of readers in an RFID network. The distribution for the probability a reader will communicate can be modeled with both uniform and exponential distributions.

5.4 Reader-level Servers

5.4.1 Overview

Directly above the readers are the reader-level servers (R-servers). R-servers are responsible for managing reader communication and relaying state information to the *Q*-server servers for processing. R-servers have more intelligence than the readers so they can better allocate time and frequency resources. Each R-server is responsible for a set of readers in the network. The total number of R-servers in the network as well as the number of readers that each R-server is responsible for, varies from 1 to *n*.

5.4.2 Collisions

In contrast to readers, R-servers are responsible for collisions avoidance for the readers within its control. An R-server knows the constraints for the readers under its control. Given the previous example of Readers A, B and C. Let s say there is one R-server A, which is responsible for the entire network of three readers. R-server A knows that in terms of interrogation zones and possible interference, the three readers are fully connected, so each can collide with the other. When the readers make a request to read, the R-server does not allocate resources to the readers that would result in a frequency collision. Additionally, the R-server may avoid tag collisions as well in allocating time slots, but at the very least, R-servers allocate resources in such a way as to guarantee that there will be no frequency collisions within its network of readers.

It is important to note that while the R-servers guarantee no collisions within its network of readers, collisions can still occur! The only scenario in which no collisions are guaranteed is with one R-server which is responsible for an entire network, as in the

previous example. This is due to border constraint violations that an R-server is not aware of. Given the same reader network of three readers A, B and C, let us now have a network with three R-servers (R-servers 1, 2 and 3), with each R-server responsible for one reader, Readers A, B and C respectively. In this case, while each R-server guarantees no collisions within its network of readers, there will still be collisions if Readers A, B and C use the same frequencies and read at the same time. The R-servers cannot directly avoid this situation. Fortunately, the *Q*-learning servers will be able to solve this problem, as will be discussed in the description of the *Q*-learning servers.

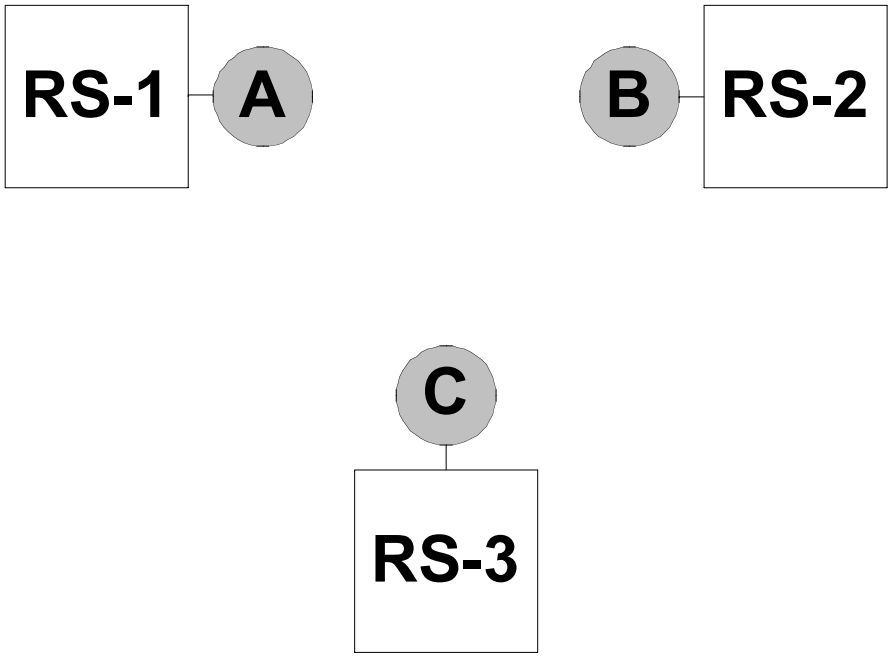

Figure 5-2. Three reader, three R-server configuration.

5.4.3 Fairness

Due to limited resources in both time and frequency, it is very likely that an R-server will not be able to service all requests by its readers during a given round. The problem that

arises from this is that there will be competition for the limited resources, creating a possible fairness issue. It is likely that those readers which make requests most often or those readers which are closest to the R-server may receive a majority of the resources, resulting in possible starvation of the readers which do not request as often or are further away.

To address the problem, a simple fairness algorithm is implemented in the R-server tier to insure that every reader gets to read during a specified period of time. Each R-server maintains a priority queue for the readers within its jurisdiction. This queue is first initialized randomly. As a reader s requests are granted, it moves to the end of the priority queue. This by itself would not effectively address the problem. This is because in all likelihood, there will be variance in read rates, and potential problems with more favorable initial queue priority. For example, let s say that the queue for Readers A, B, and C is initialized to be {B, C, A}. If all three readers happen to make read requests every round, or if they happen to make requests during the same round, the B will always have priority. To prevent this from happening, the queue is periodically repopulated randomly.

Using the same network of three readers, here s a simple example to that illustrates how the algorithm works. Assume that R-server 1 is control of all three readers and it only has one frequency with one time slot to read. In terms of request rates, Reader A makes a request every round, Reader B makes a request roughly every 10 rounds, and Reader C makes a request roughly every 15 rounds. After five rounds, the priority queue will be {C, B, A).

In the next round, Reader A is again the only reader making a request. While C and B have priority over A, they do not make requests, so the resources are again allocated to Reader A. In round 10, Reader B will be given the frequency and the time slot. The next round, the priority queue will become {C, B, A}. In round 16, the queue is {B, A, C}.

In the next four rounds, there is a significant traffic increase, so all three readers make requests every round. In round 20, the queue is randomly repopulated to {C, A, B}.

While the queue normally proceeds in a deterministic manner, the periodic randomizations ensure fairness among those with similar traffic loads.

5.5 *Q*-learning Servers

5.5.1 Overview

Q-learning servers comprise the highest tier in the network hierarchy. Unlike the reader and R-server tiers, the *Q*-server level can also be arranged hierarchically. The highest tier of *Q*-servers is the root tier. The structure of the network is such that there can be single to multiple root *Q*-servers. *Q*-servers have the most intelligence and using the *Q*-learning algorithm, are designed to optimize time and frequency resources to the tiers directly below.

5.5.2 Resource Allocation

Each *Q*-server has global knowledge of the resources available in the network. This is to insure that every *Q*-server assigns the same time slots throughout the system. This

simplifies the tracking of when readers are communicating and makes the allocation of time slots more organized and manageable.

In terms of available resources, only *Q*-servers at the root *Q*-server tier can allocate all the resources. If there are multiple *Q*-server tiers, the *Q*-servers below the root in the hierarchy must wait to be allocated resources before they can allocate these same resources to the tiers below them.

To simplify communication in the system, resource allocation is done in rounds. The length of a round is based on the tier in the hierarchy. The root *Q*-server tier has the longest round, and each tier below it has progressively shorter rounds. This allows each tier to accumulate state cost information over time.

At the start of each round, a *Q*-server pings all the servers below it (Q or R-servers). This ping is propagated down to the reader tier, where readers begin making requests for frequencies and time slots. Because none of the R-servers above the readers have resources allocated to them yet, the first round is essentially a throwaway round.

After the initial round, the root *Q*-server begins allocating resources to the servers below. The fairness queuing algorithm used at the R-server tier is employed at the *Q*-server tier as well. When a *Q*-server receives a request for a resource, it can execute one of three actions. It can

1) ADD a resource
2) SWAP resources
3) REMOVE a resource

The *Q*-server determines which action to take based on predefined thresholds for frequency and tag collision rates, as well as rejection rates. Resources are allocated or

taken away from servers based on whether their performance levels are above or below the thresholds. If a server making a request is experiencing a high level of rejections, the *Q*-server will most likely add a resource. If a server is performing well, a *Q*-server may remove a resource from that server. Removes are used to minimize the number of resources used in the system. Alternatively, a *Q*-server may choose to SWAP resources. If a server is not rejecting many requests and has fluctuating collision levels near the thresholds, then the *Q*-server will likely perform resource swaps to optimize the server s performance. These three actions are done probabilistically, based on performance.

5.5.3 States

To aid the *Q*-learning process, all servers (*Q*-servers and R-servers) keep track of their states over time. These states consist of vital information about the performance of the tier below in the hierarchy. Some of the information contained in these states is used to determine the cost of an action taken by the *Q*-learning agent. For scalability, this recorded information is the same for both R-server and *Q*-servers, the only thing that changes is the scope of the information because each tier in the hierarchy is redundant.

The following is a table of the information fields contained in each server state:

Field	Description
Server Identification Number	The unique number of the server which identifies it to the rest of the network for communication.
Requests	The number of requests that have been made in the tiers below.
Grants	The number of granted resource requests in the tiers below.
Rejections	The number of rejected requests in the tiers below.
Tag Collisions	The ~~the~~ number of tag collisions experienced in the tiers below.
Frequency Collisions	The number of frequency collisions experienced in the tier below.
Tag Interferers	The number of tag collision instances experienced in the tiers below.
Frequency Interferers	The number of frequency collision instances experienced in the tiers below.
TTL (time to live)	The amount of time a reader has to read when it is granted a frequency and a time slot. This enables global uniformity of time slots.
Time Slots	The list of the time slots that have been allocated to server.
Frequencies	The list of frequencies that have been allocated to a server.

Table 5-1. Server state information.

After each round, the servers update all the categories of the state. After a certain number of rounds, specified by the server directly above it, the server sends a state update to the server directly above it with all the data that it has accumulated over the rounds. The server then waits a predetermined amount of time for a response from the higher server, indicating whether or not the server will be allocated new resources. During this time, the server adds the state information to its history, and resets its current state.

5.5.4 Minimizing the Cost Function

The cost function is essential to the success of the *Q*-learning agent. The cost function used is the one described in Chapter 3 and is the same at each tier in the *Q*-learning hierarchy. However, because of the varying control scopes of each *Q*-server, the input data will be different.

As outlined previously in the *Q*-learning algorithm, *Q*-servers attempt to allocate resources to the tiers below in such a way as to minimize the cost function. After the values of the *Q*-table have been updated using the discounted cost from the latest state information, the *Q*-server tries to determine what the optimum actions are. The *Q*-server follows a control policy which seeks to assign actions with the minimal *Q*-values. However, to avoid converging to local optima through greedy resource assignment, the *Q*-servers employ some randomness to determine whether to execute the optimum action.

The likelihood of staying in the prior state or transitioning to the next state is determined by an exploration rate which is dependent on the difference between the value of the previous action and the value of the minimum action. First we define ε

$$\varepsilon = \frac{(Q_{prev} + Q_{\min})}{Q_{\min}} \quad (26)$$

where ε is the relative value of the previous *Q*-value and the minimum *Q*-value. Additionally, we define a change rate Δr

$$\Delta r = \frac{\#\,of\ changes\ made}{total\ \#\,of\ nodes} \quad (27)$$

This change rate determines the likelihood that a change in state will be made, based on ho many changes have already been made in a round. The probability that the *Q*-server

will assign the previous action or move to a new state is given by an exploration rate β, given by the weighted sums of ε and Δr

$$\beta = [(n) * \varepsilon] + [(1 - n) * \Delta r] \quad (28)$$

where n is between zero and one and a collision rate λ. λ is defined as

$$\lambda = \frac{\#of\ collisions\ for\ i}{total\ collisions} \quad (29)$$

where i is the node number and the total collisions represent those in the tiers below the Q-server. The collision rate is discounted by a change factor f, which further determines the likelihood of a state change. The Q-server generates two random numbers between zero and one. If the first is less than the exploration rate and the second is less than the collision rate, the Q-server chooses the action with the minimum Q-value. Otherwise, it repeats the previous action. By following a policy that does not always choose the minimum Q-value, the algorithm is less likely to converge to local optima.

Additionally, if the minimum action is selected, it is not necessarily performed. With some probability p based on the weighted collision rate $f * \lambda$, the Q-server will perform a random action to further explore the state space. It is important to note that this weighted collision rate is decayed over time in a similar manner as the learning rate α.

Chapter 6

Simulation Results

To evaluate the performance of the HiQ algorithm, a series of experimental simulations were run in a simulated reader network. By varying various parameters including the size of the network, the hierarchical granularity, the availability of resources and the traffic density, the simulations demonstrated the versatility and adaptability of the HiQ algorithm. We present the results of six simulation sets and discuss HiQ algorithm s performance in each of them.

6.1 Overview

The HiQ simulations were developed in Java using the SimJava [11] simulation package. We present six sets of experimental simulations that illustrate HiQ s performance with for a range of parameter values. The first five sets of experiments were performed on variations of a simple square grid topology. The final set of experiments was simulated on a FAP example with real-world interference constraints.

6.1.1 Square Grid Topology

As previously mentioned, we can model a reader network as a connected graph. For the experiments, we simulate the performance of HiQ on a network topology based on a fully-connected square graph, shown in figure 6-1.

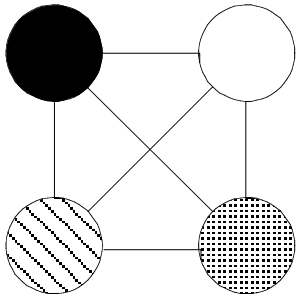

Figure 6-1. Simple 4-coloring of a square grid topology.

This graph is the basic building block for larger square grids, which can effectively model a system of readers. Each vertex in the graph represents a reader while each edge connecting these vertices represents interference patterns. In a network of readers, neighboring readers who may collide with each other need to use different frequencies and/or time slots. The allocation of frequencies and time slots to readers is analogous to coloring each vertex in a graph. If two adjacent vertices do not share the same color, the color scheme satisfies the edge constraints. Similarly, if two readers are communicating using different frequencies or time slots, they will not collide.

In the square grid topology, the minimum number of colors required to satisfy the edge constraints is four. The topology is useful because it is easy replicated and even the largest square grids can still be 4-colored. The simplicity and scalability of this topology make it well-suited for modeling reader networks. We now present the results of the HiQ simulations.

6.2 Experimental Results and Discussion

The goal of the first set of experiments was to test the scalability of the HiQ algorithm. The next two sets of experiments tested HiQ s performance with different R-server and

Q-server granularities and hierarchies respectively. The last two sets of simulations performed on the square grid simulated HiQ s performance given various resource availabilities and the algorithm s ability to adapt to different traffic patterns from the readers. The final experiment was performed on a topology taken from a real-world instance of the Frequency Assignment Problem.

For each experiment, we present relevant results and regarding tag collision rates, frequency collision rates, blocking rates (rate of rejected requests) and the number of distinct resources HiQ used and discuss their significance. The tag and frequency collision rates are found by simply dividing the total number of tag and frequency collisions, respectively, by the total number of resource grants. The blocking rate is found by taking the total number of rejections and dividing it by the total number of requests. The number of resources used is the number of frequency and time slot pairs that HiQ used by the end of the simulation. Each pair is the combination of one frequency and one time slot. For example, in a system with two frequencies and three time slots, there are a total of six pairs translating to six distinct resources.

The simulations of 10000 iterations were run. With the exception of the experiments testing traffic distributions, readers make communication requests 100% of the time. That is, when a reader is pinged by its R-server, it will always request a frequency and a time slot.

6.2.1 Scalability

The first aspect of HiQ that was tested was its ability to perform in both small and large networks. Using the square grid topology, the performance of HiQ in networks

consisting of 9, 16, 25, 36 and 49 readers was simulated. Each reader is controlled by one R-server, and there is a single Q-server controlling the R-servers. In each experiment, there are a total of four frequencies and four time slots available for allocation. The tag collision rate, frequency collision rate and blocking rate over time are presented in Figures 6-2 through 6-4.

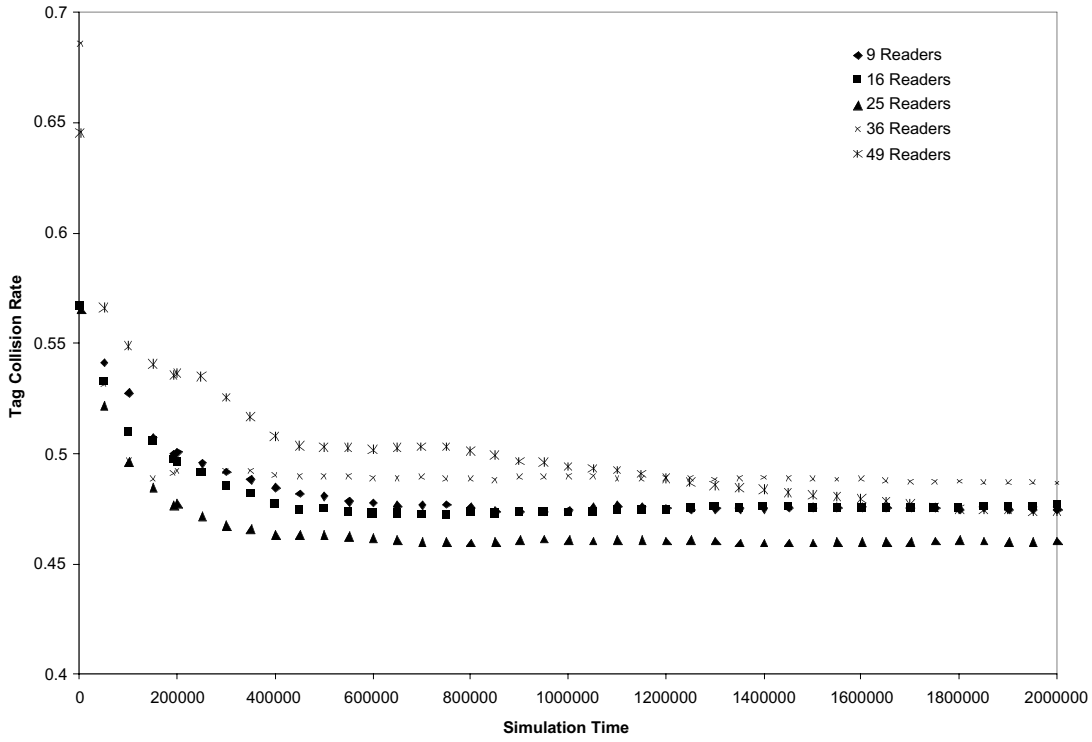

Figure 6-2. Tag collision rates for varying number of readers.

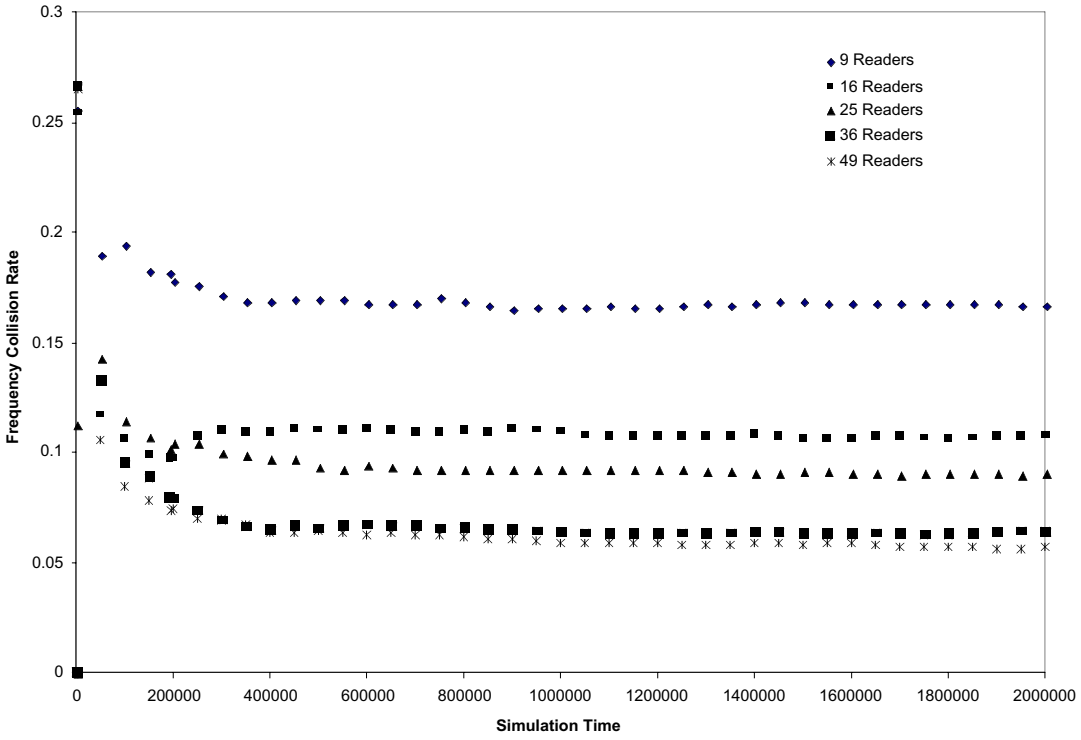

Figure 6-3. Frequency collision rates for varying number of readers.

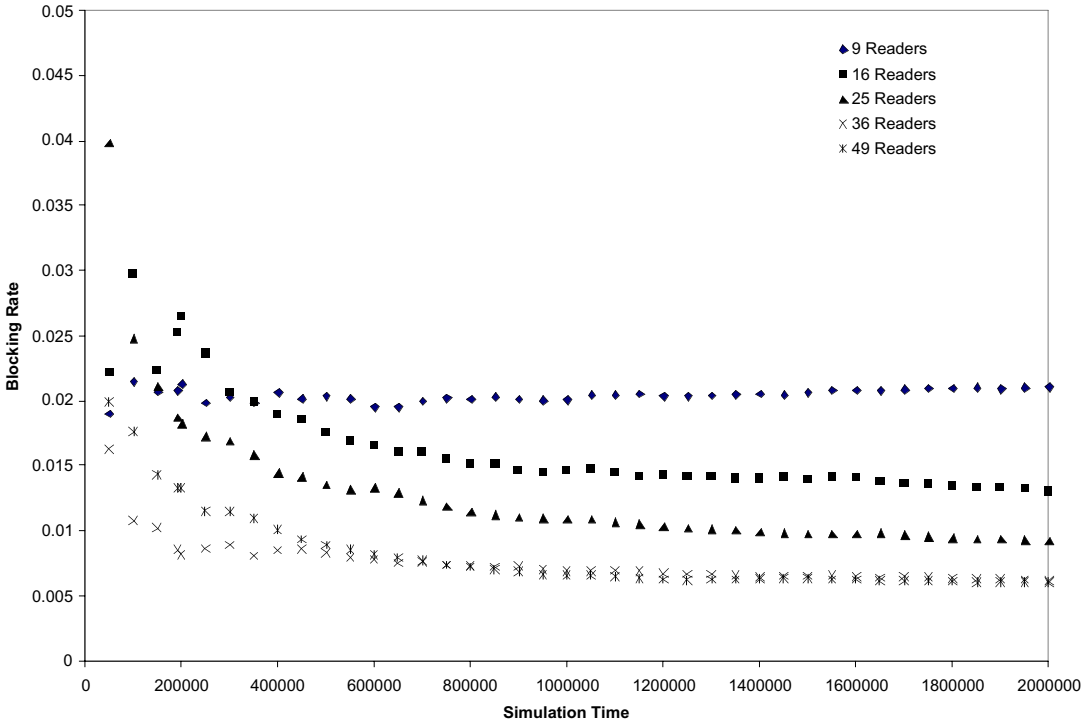

Figure 6-4. Blocking rates for varying number of readers.

R	RS	RS/R	QS	QS T	RS/QS	RRR	TS	Freq	Resources
9	9	1	1	1	9	100	4	4	7
16	16	1	1	1	16	100	4	4	10
25	25	1	1	1	25	100	4	4	12
36	36	1	1	1	36	100	4	4	14
49	49	1	1	1	49	100	4	4	15

Table 6-1. Number of resources used by HiQ for varying number of readers.

Key:
Experiment: 1 = scalability, 2 = R-server granularity, 3 = *Q*-server hierarchy, 4 = resource availability, 5 = traffic distributions; *R* = # of readers, *RS* = # of R-servers, *R/RS* = # of readers per R-server, *QS* = # of *Q*-servers, *QS-T* = # of *Q*-server tiers, *RS/Q* = # of R-servers per *Q*-server, *RRR* = reader request rate, *TS* = # of available time slots, *Freq* = # of available frequencies, *Resources* = # of distinct time slots & frequencies used

The simulation results indicate that given abundant resources, HiQ s performance scales well with the size of the network. The frequency collision rates and the blocking rates in the largest network, with 49 readers, were similar to those of in the smallest network, with just nine readers. In fact, the larger networks had lower frequency and blocking rates than their smaller counterparts. Looking at ~~Table ?~~Table?, it is clear why, since HiQ only allocates a larger number of resources when it is needed. HiQ allocates only seven resources in the nine reader network while in the 49 reader network, it allocates fifteen. This is likely due to the increased exploration of resource availabilities in the larger network of readers. Whereas the smaller networks converge relatively smoothly and quickly, the larger networks undergo a slightly erratic and slower convergence because the Q-server must allocate many more resources.

From this first set of experiments, the tradeoff between request blocks, frequency collisions and tag collisions can already be seen. As performance measures, we have placed the greatest cost on the frequency and rejection collisions because successful

reader communication is highly unlikely and not possible, respectively. When a reader experiences a tag collision, it cannot determine whether or not the collision has actually affected its communication with tags in its interrogation zone. Therefore, we constructed an ordered cost structure where the relative weights of frequency collisions and request blocks are the most costly occurrences, with rejections slightly worse than frequency collisions. Consequently, HiQ allows for significantly greater rates of tag collisions, since they are not weighted as heavily in the cost structure.

6.2.2 R-server Granularity

The second set of simulations experiments with different R-server configurations. The simulations were run on a square grid consisting of 25 readers, with each R-server controlling as few as one reader and as many as nine readers. This experiment tests the effectiveness of the simplistic greedy algorithm run at the R-server tier, and how *Q*-learning handles different granularities of R-servers below. As in the first set of simulations, there are four frequencies and four time slots available for allocation. The results of these experiments are presented in Figures 6-5 to 6-7.

It is important to note that when there is one R-server per reader, then an R-server should not be allocated more than one resource. If a certain assignment results in a high cost, the resource should be swapped or removed by the *Q*-servers. This is because R-servers assign frequencies and time slots in a simple, deterministic manner in, leaving them poorly equipped for choosing an optimum frequency and time slot out of a set of available frequencies and time slots. When an R-server is responsible for more than one reader, then the *Q*-server gives the R-server more resources by doing periodic adds, based

on the performance of the readers that R-server controls. A *Q*-server limits the number of resources it allocates such that the learning process and optimization continues. This is to prevent R-servers from holding on to more resources than it uses.

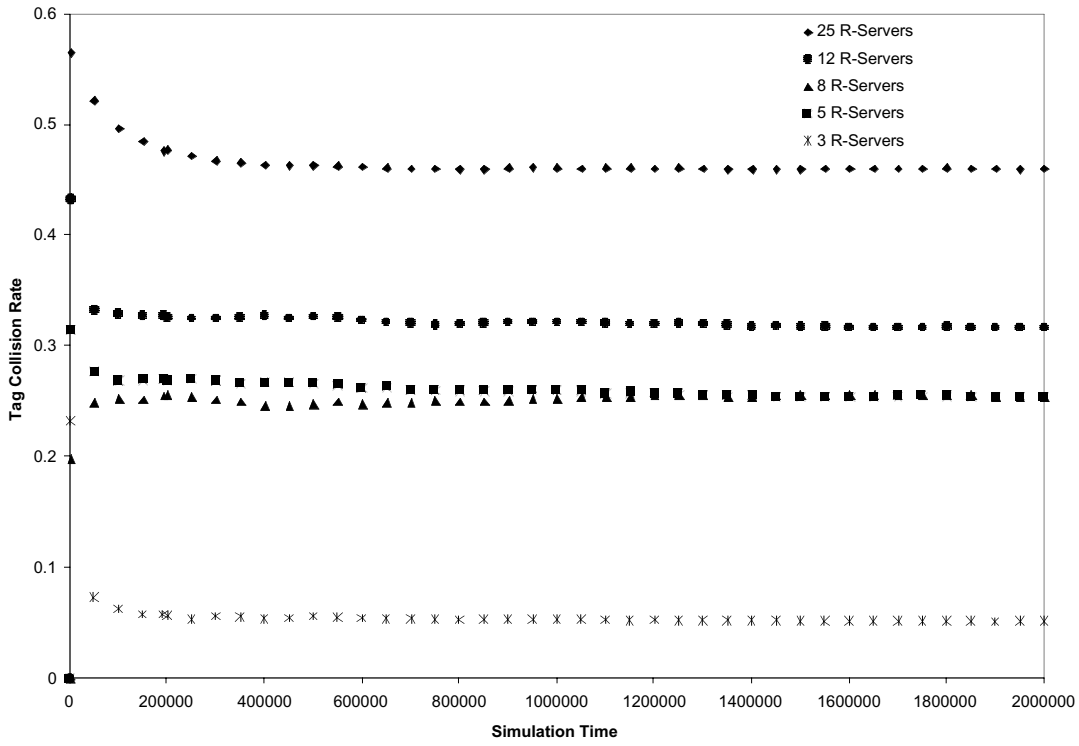

Figure 6-5. Tag collision rates for varying number of R-servers.

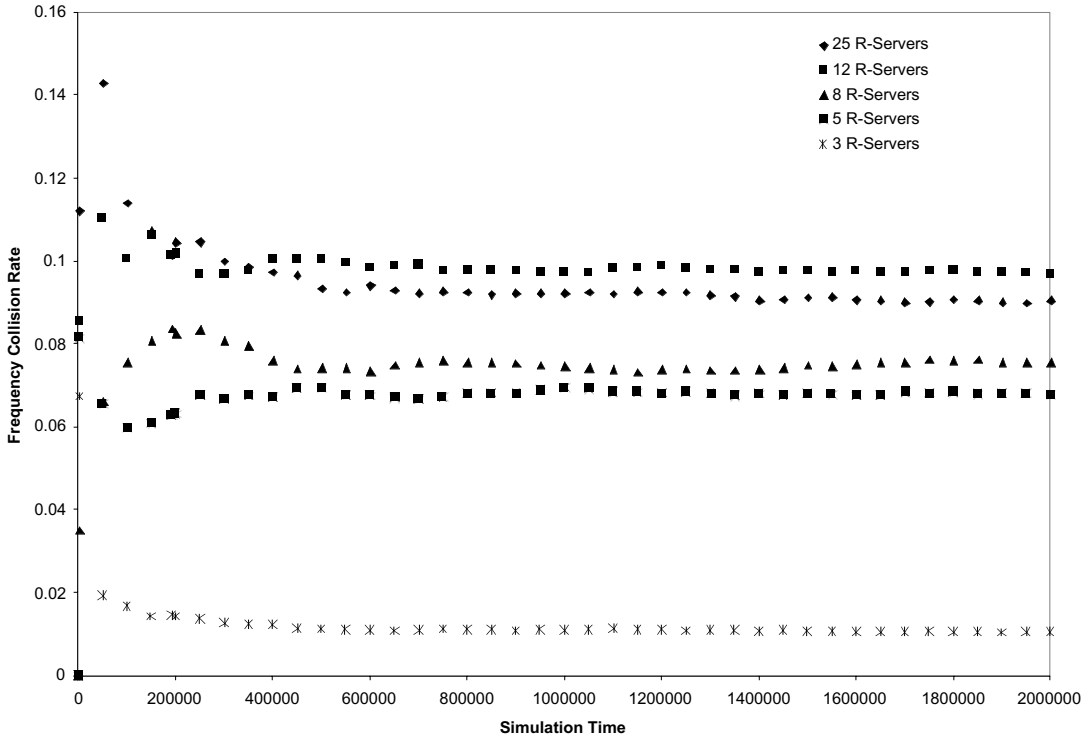

Figure 6-6. Frequency collision rates for varying number of R-servers.

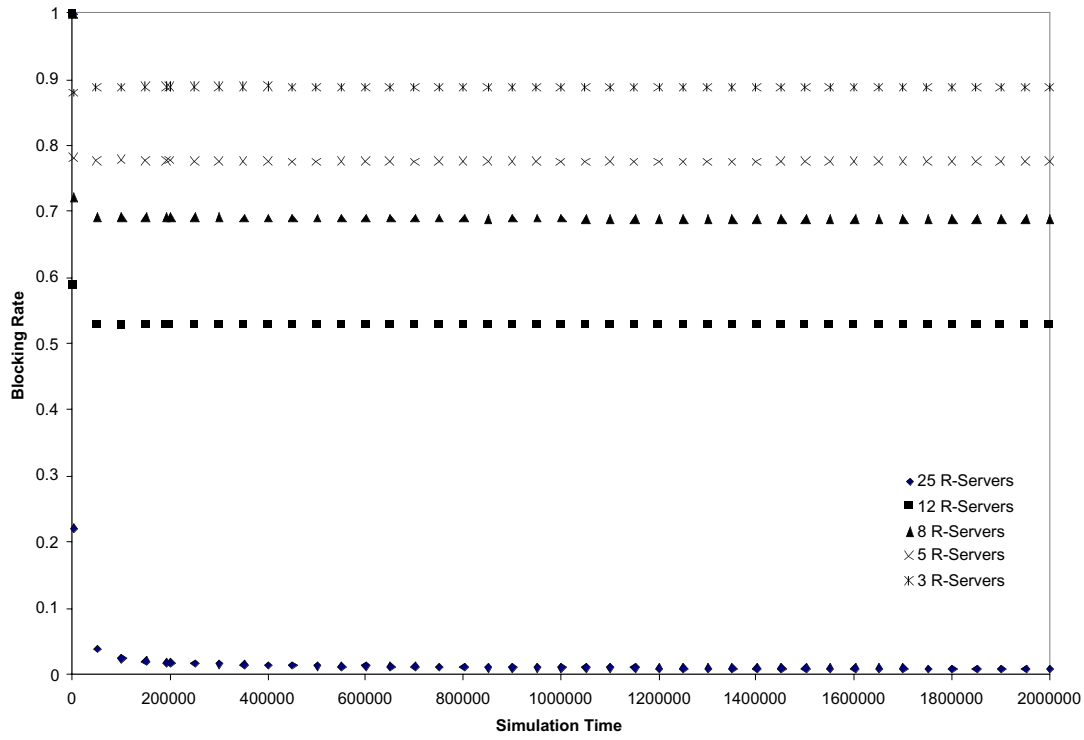

Figure 6-7. Blocking rates for varying number of R-servers.

R	RS	RS/R	QS	QS T	RS/ QS	RRR	TS	Freq	Resources
25	25	1	1	1	25	100	4	4	12
25	12	2	1	1	12	100	4	4	8
25	8	3	1	1	8	100	4	4	6
25	5	5	1	1	5	100	4	4	4
25	3	8	1	1	3	100	4	4	3

Table 6-2. Number of resource used by HiQ for varying number of R-servers

Although the number of time and frequency resources available for HiQ allocate is still abundant, a system with a great deal of R-server control — where a single R-server controls many readers — may underutilize these resources. While greater R-server control limits the number of frequency collisions, less information is passed back to the *Q*-server about how well the tiers below it are performing. The allocation policies the learning agent tries to follow are based on the performance feedback it receives from the R-servers. Since these policies are determined based on what allocation scheme the *Q*-server s learning agent has previously allocated, the performance information must correlate these assignments. With greater R-server control, many of these correlations are lost because the decisions the learning agent makes are not synchronized with the actual state of the network. Fundamentally, the learning agent is exploring the state space based on severely incomplete information, leaving the R-server to act as the main decision maker in terms of resource allocation. Because the R-server is very simple — its only goal is to prevent frequency collisions — the complexity and learning of the HiQ algorithm is reduced to the capabilities of the R-servers.

The underutilization of resources with fewer R-servers results in poor performance in terms of blocking rates. In the 3 R-server network only three resources are allocated while in the 25 R-server network, twelve are allocated. With fewer R-

servers and resources, the tag and frequency collision rates are lower. However, the higher blocking rates for these configurations outweigh the gains in collision avoidance. For a network with one *Q*-server and 3 R-servers, with each R-server controlling eight or nine readers, the blocking rate is close to 90%, with less than 2% frequency collisions while the 25 R-server network has a less than 2% blocking rate with roughly 10% frequency collisions. This shows that tradeoff between blocking rate and collision rate with fewer R-servers is significantly greater than that same tradeoff with more R-servers.

6.2.3 *Q*-server Hierarchy

The simulations run to test the *Q*-server are similar to those run for the R-servers. Here, we experiment with multiple *Q*-servers, as well as multiple *Q*-server tiers. The previously mentioned guidelines for *adds, swaps,* and *removes* become more important when multiple *Q*-servers and *Q*-server tiers are present. Which action the algorithm chooses to perform is based on some predefined probability, which is decayed over time so that the system can converge.

Experiments were performed with two *Q*-server tiers. At the highest tier *Q*-server tier, there is a single *Q*-server, the *root Q*-server. The number of *Q*-servers in the tier between the root and the R-server tier was varied between three and twelve *Q*-servers in a network of 25 readers and 25 corresponding R-servers. Again, there are four frequencies and four time slots. The results are shown in Figures 6-8 to 6-10.

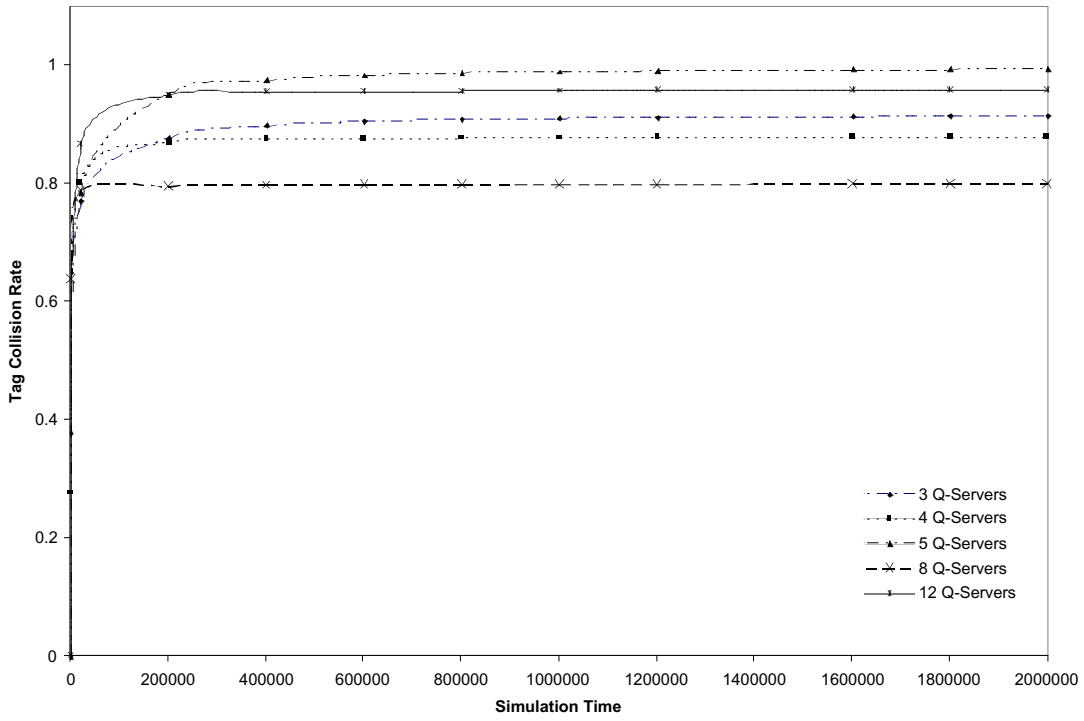

Figure 6-8. Tag collision rates for two *Q*-server tiers and varying number of *Q*-servers

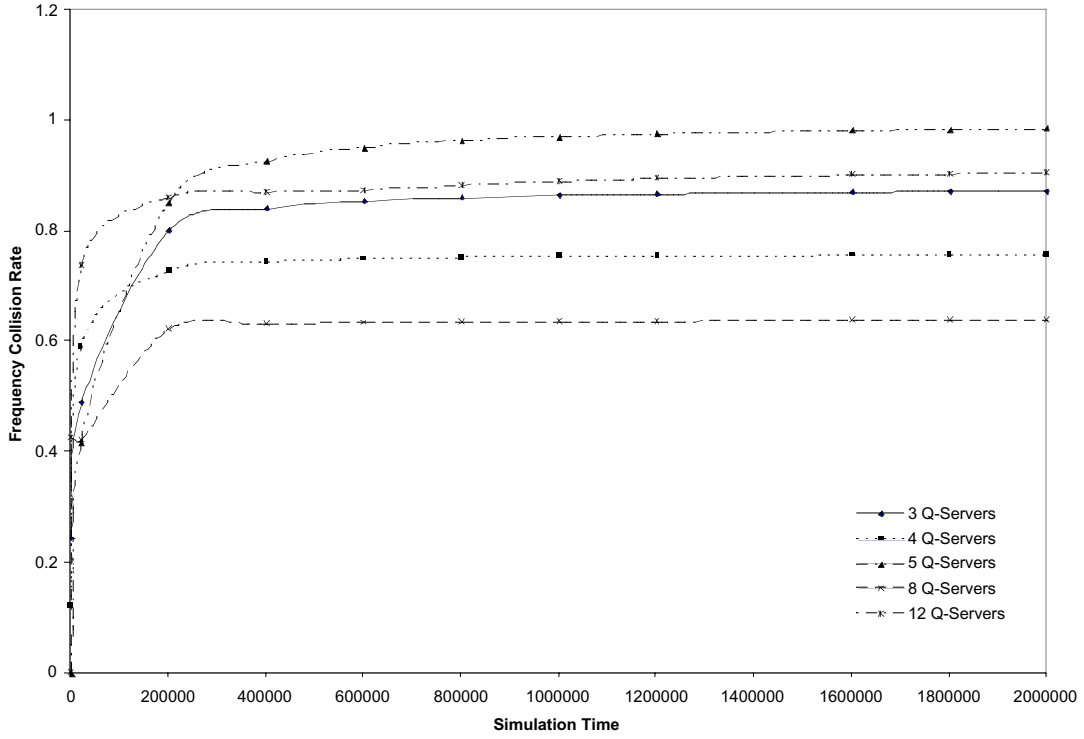

Figure 6-9. Frequency collision rates for two *Q*-server tiers and varying number of *Q*-servers.

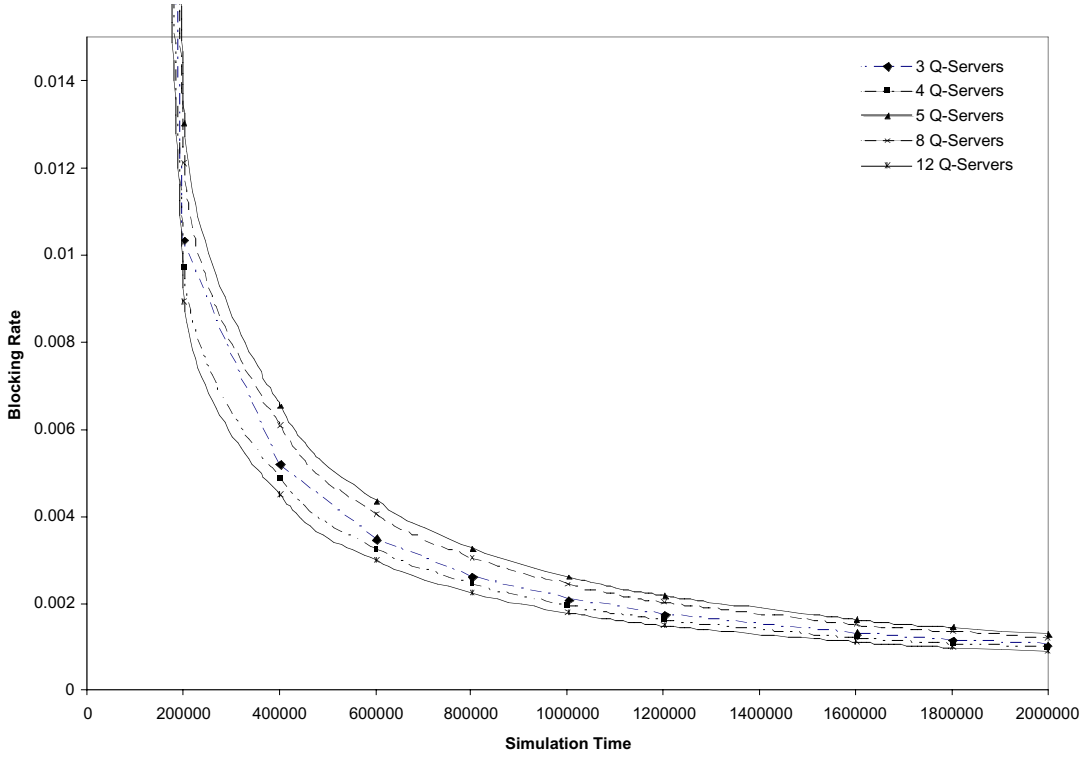

Figure 6-10. Blocking rates for two *Q*-server tiers and varying number of *Q*-servers.

R	RS	RS/R	QS	QS T	RS/ QS	RRR	TS	Freq	Resources
25	25	1	3	2	8	100	4	4	3
25	25	1	4	2	6	100	4	4	4
25	25	1	5	2	5	100	4	4	5
25	25	1	8	2	3	100	4	4	6
25	25	1	12	2	2	100	4	4	8

Table 6-3. Number of resources used by HiQ for varying number of *Q*-servers.

While the blocking rate of the hierarchically structured *Q*-servers converges to near zero-probability, both the frequency and tag collision rates converge closer to one. This is most likely caused by the brief length of the initial exploration period.

In each of the experiments, there is an unstable period of two to four thousand iterations where HiQ extensively searches the state space. Over time, the scope of this

search is narrowed, and ultimately converges to a near-static assignment. With multiple *Q*-server tiers, the relative length of this exploration period could decrease by an order of magnitude for each additional tier of *Q*-servers. In other words, the period of non-deterministic exploration of the state space to populate the *Q*-tables is cut short. The root *Q*-server experiences the worst performance hit as a result of this shorter exploration period, causing the system to converge to assignments with high frequency and tag collision rates.

The number of resources used increases with the total number of *Q*-servers. This is similar to the behavior of the R-servers in the second experiment. However, the benefits of having more *Q*-servers is a lower blocking rate while the benefit of having more R-servers is lower frequency and tag collision rates.

6.2.4 Resource Availability

The fourth parameter that was changed was the number of available resources in the network. With a nine reader square grid, various combinations of available frequencies and time slots were tested. The results of these simulations are shown in Figures 6-11 to 6-20.

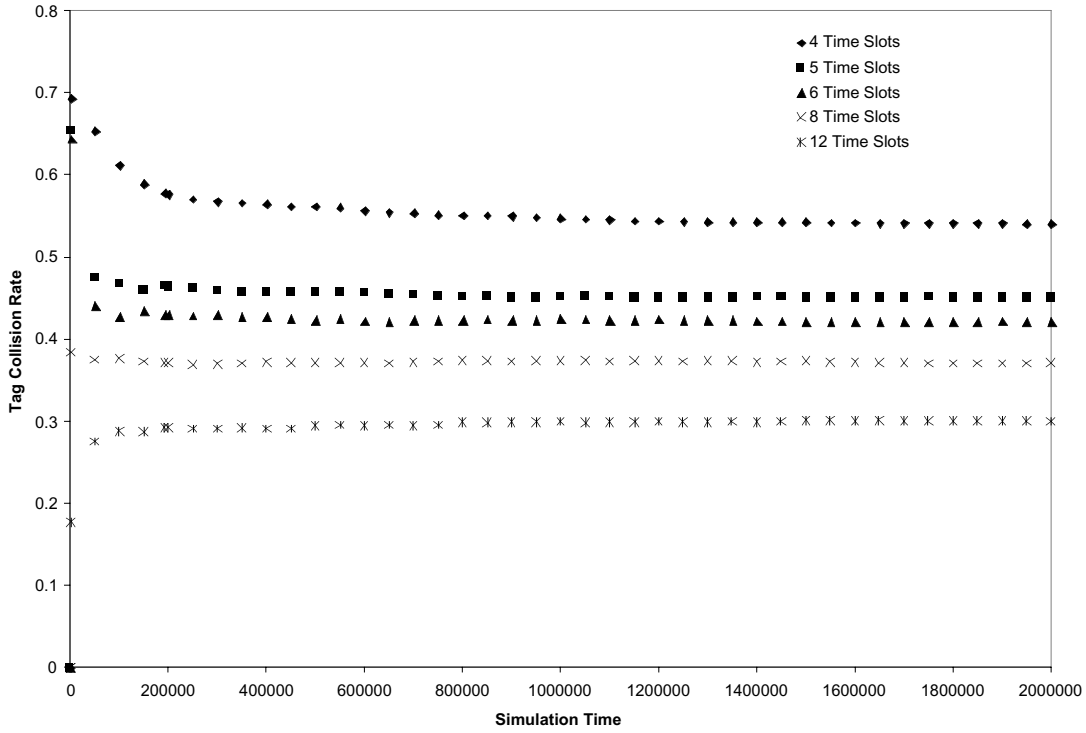

Figure 6-11. Tag collision rates for one frequency and varying number of time slots.

The above graph indicates how well the HiQ allocates time slots when there are a small, fixed number of frequencies. In the case of one frequency, the algorithm performs significantly better with an abundance of time slots, rather than a minimal number of time slots. Comparing the tag collision rate with one fixed frequency with the frequency collision rate of a system with one fixed time slot (Figure 6-14), we see that the HiQ algorithm more effectively allocates frequencies than time slots. This again goes back to the preference of tag collisions over frequency collisions and request blocks.

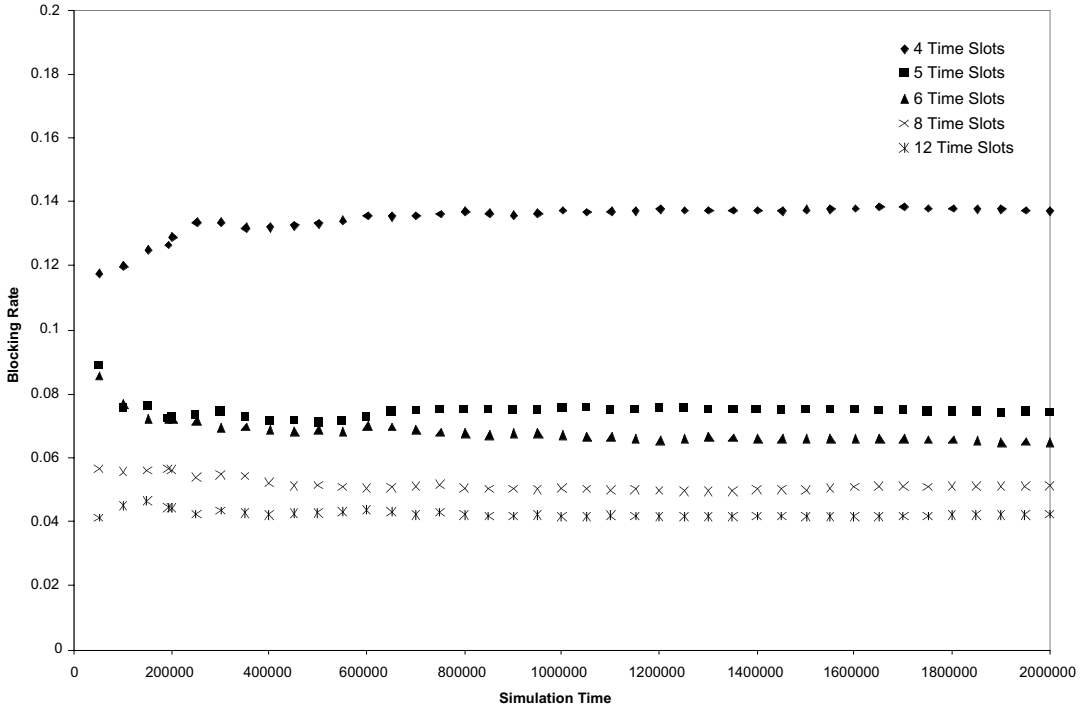

Figure 6-12. Blocking rates for one frequency and varying number of time slots.

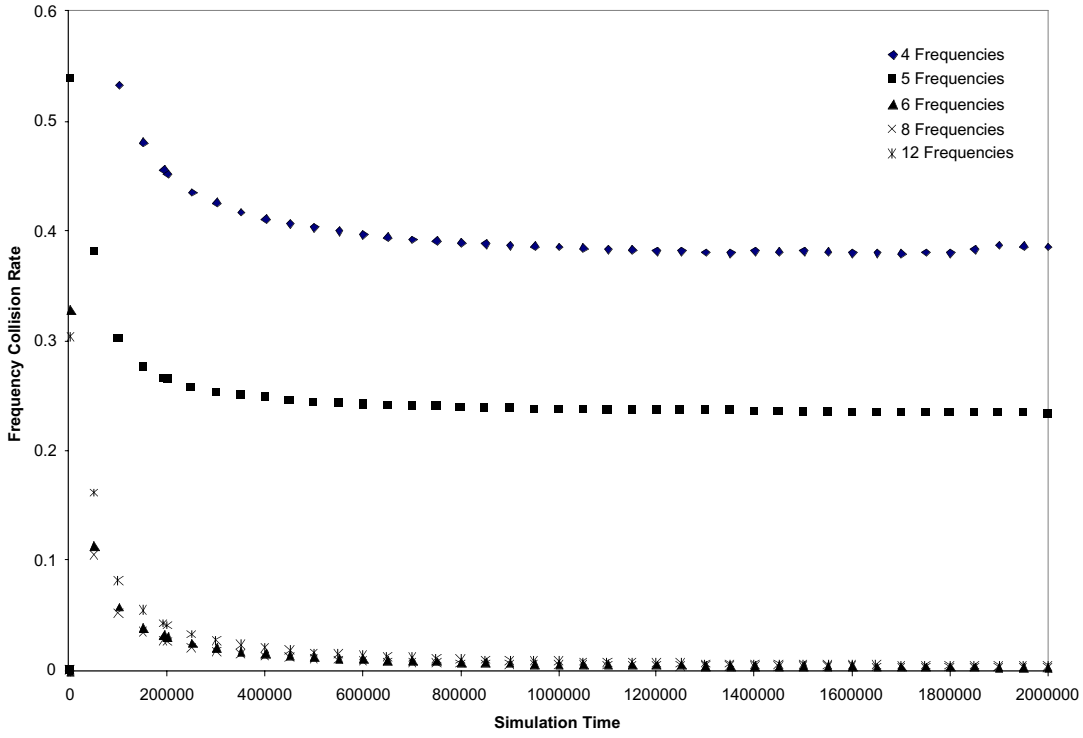

Figure 6-13. Frequency collision rates: one time slot, varying number of frequencies.

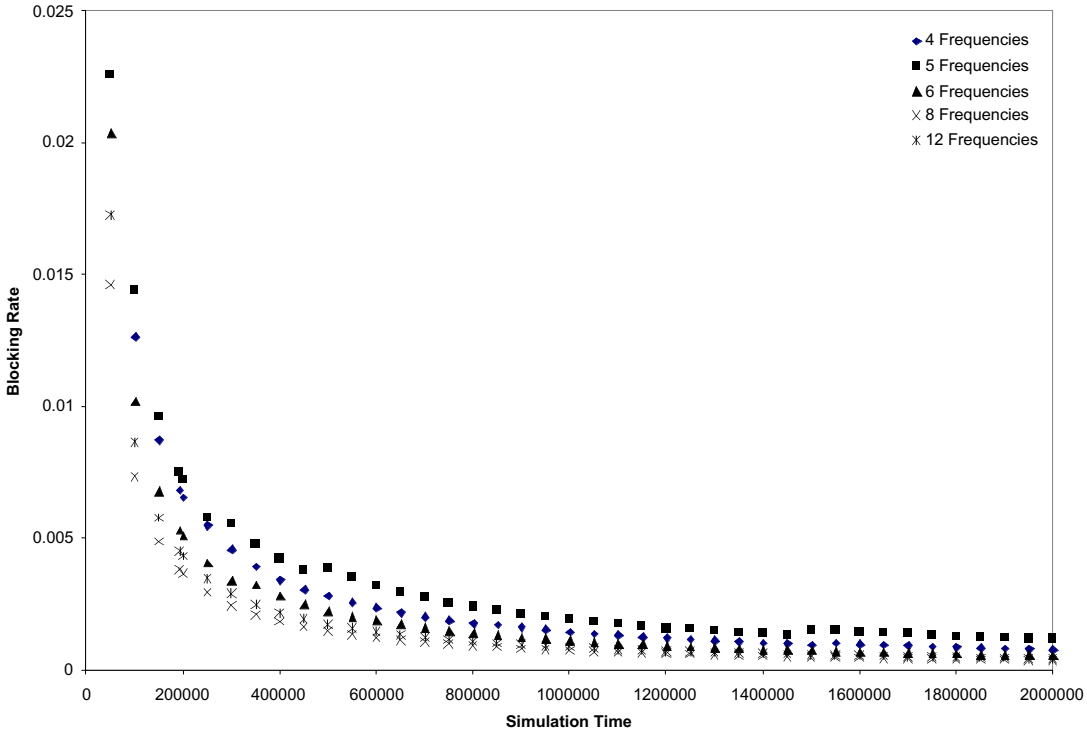

Figure 6-14. Blocking rates for one time slot and varying number of frequencies.

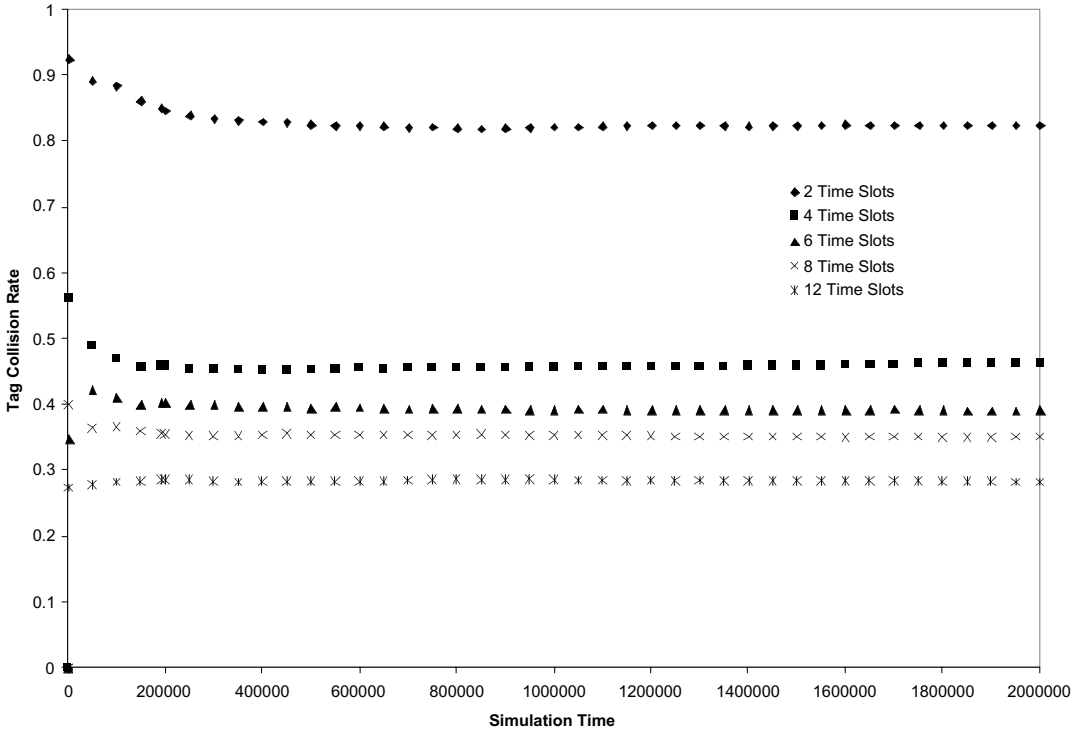

Figure 6-15. Tag collision rates for two frequencies and varying number of time slots.

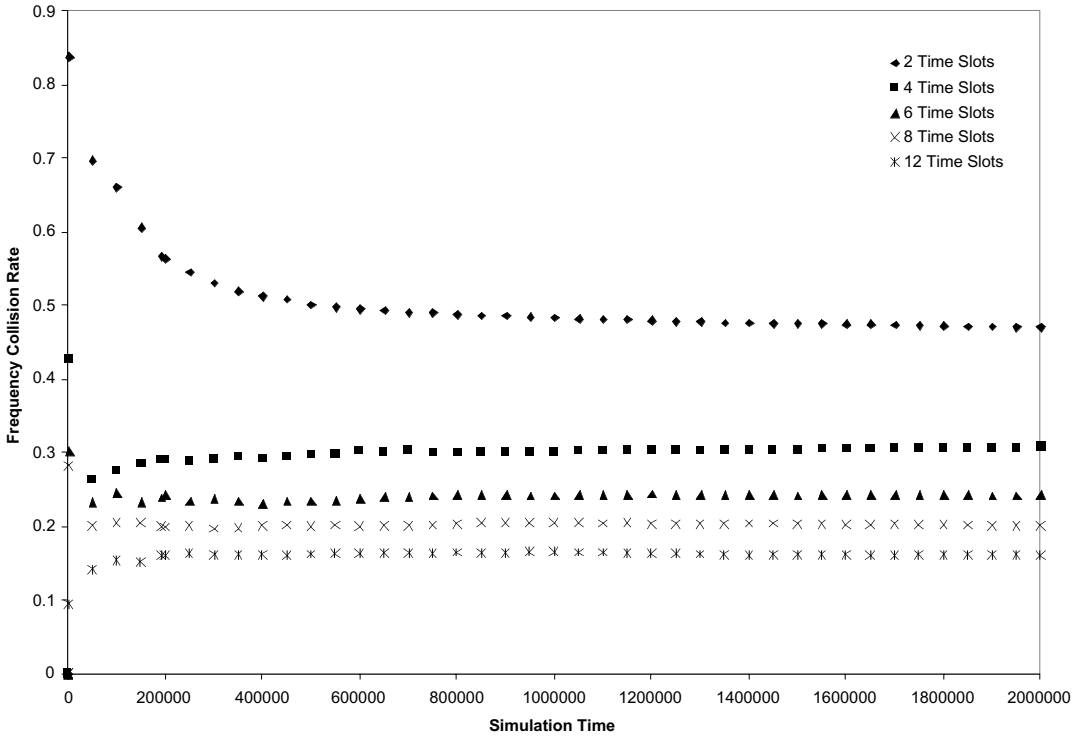

Figure 6-16. Frequency collision rates for two frequencies and varying number of time slots.

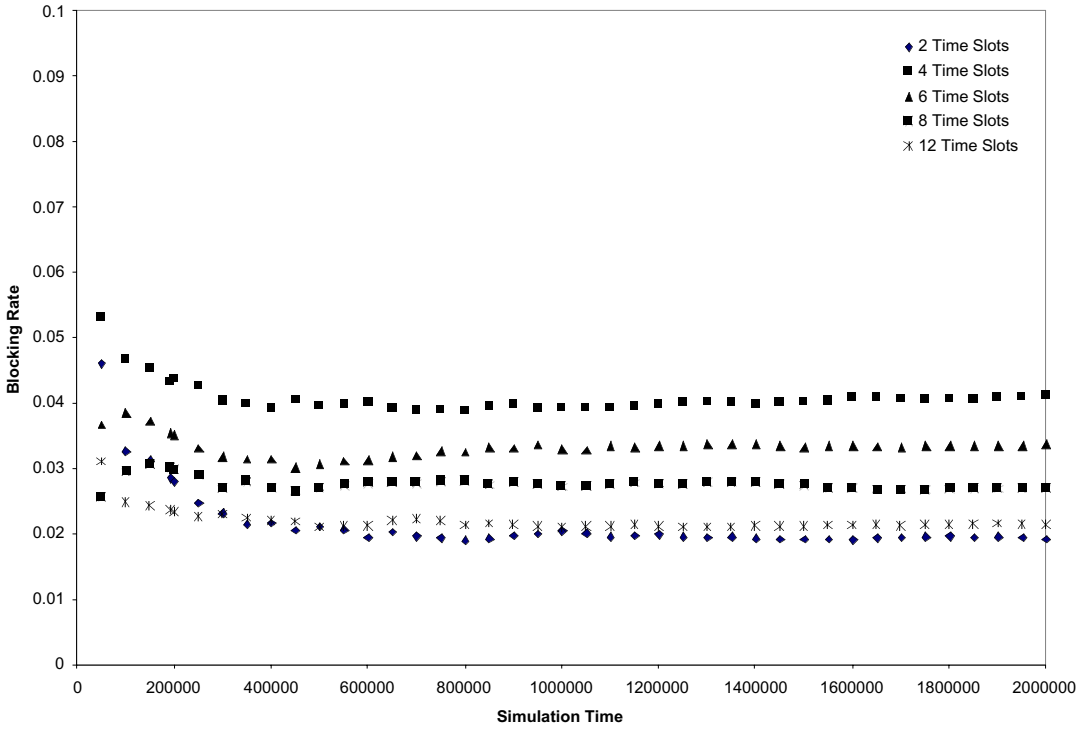

Figure 6-17. Blocking rates for two frequencies and varying number of time slots.

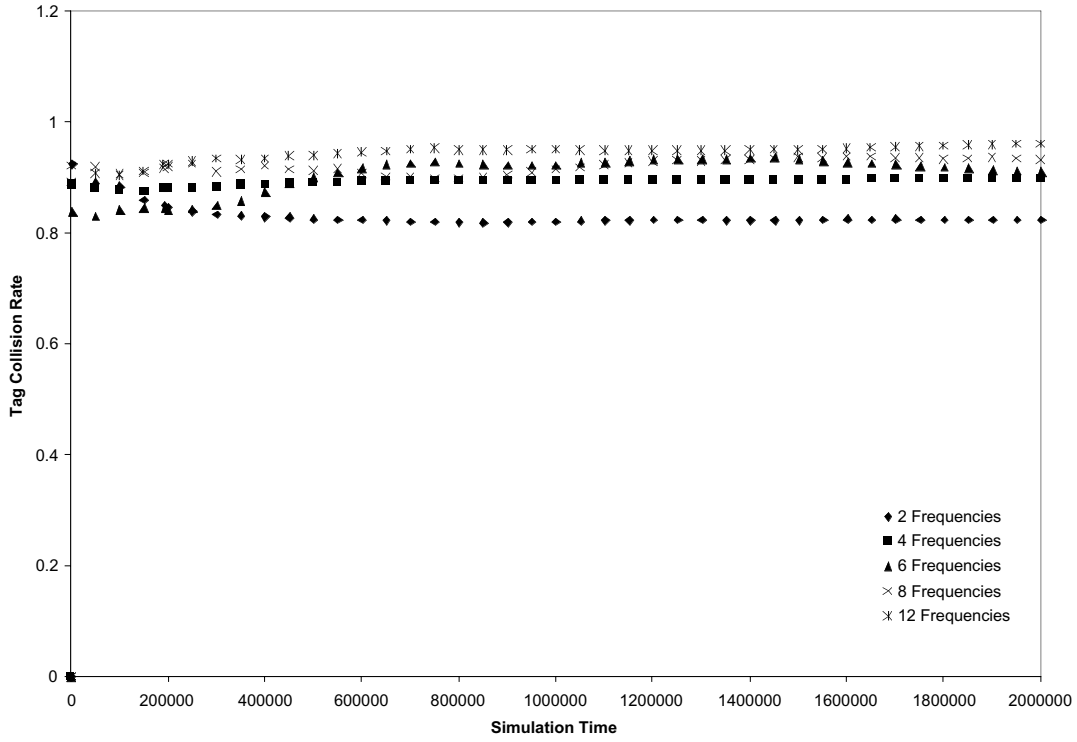

Figure 6-18. Tag collision rates for two time slots and varying number of frequencies.

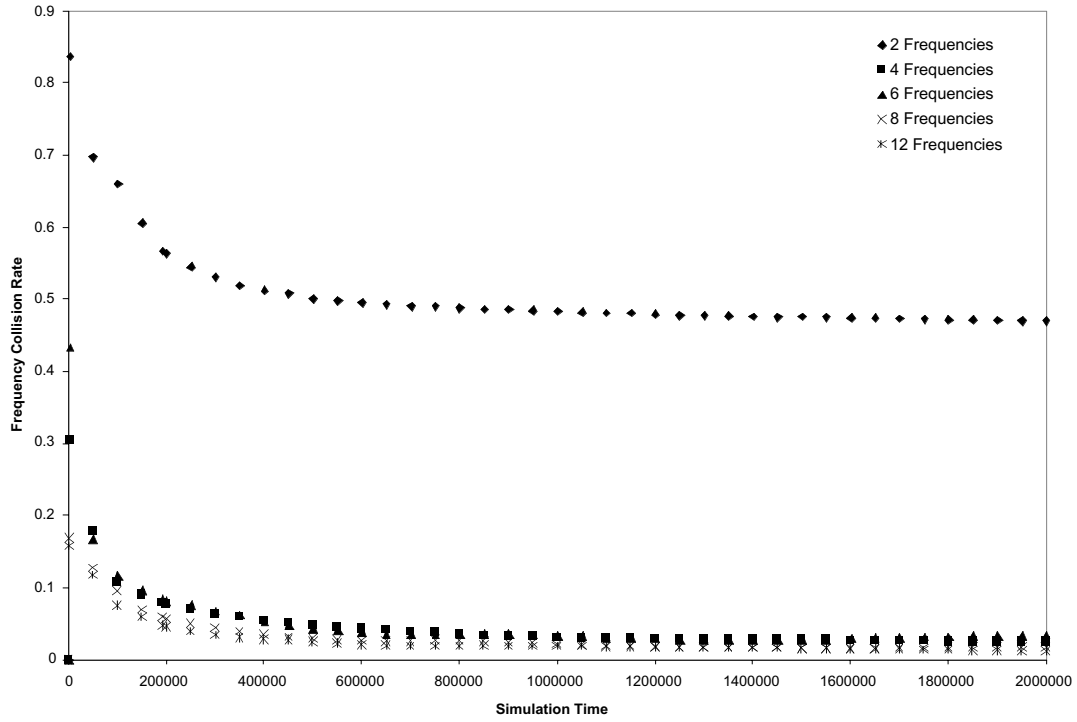

Figure 6-19. Frequency collision rates for two time slots and varying number of frequencies.

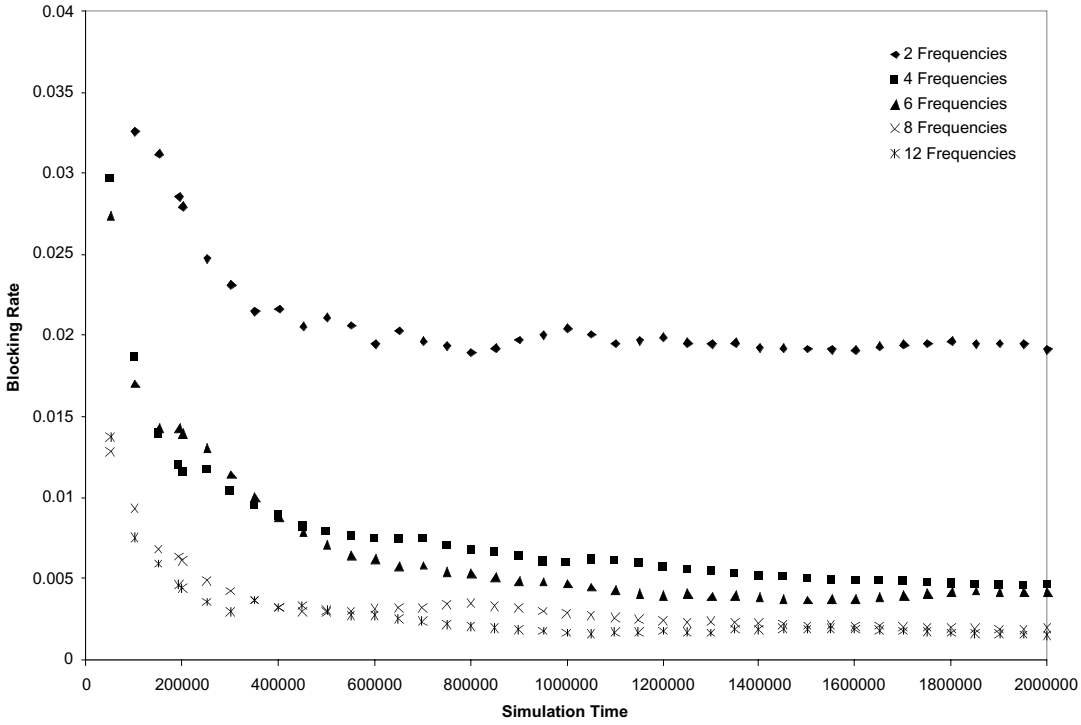

Figure 6-20. Blocking rates for two time slots and varying number of frequencies.

R	RS	RS/R	QS	QS T	RS/ QS	RRR	TS	Freq	Resources
9	9	1	1	1	9	100	4	1	4
9	9	1	1	1	9	100	5	1	4
9	9	1	1	1	9	100	6	1	4
9	9	1	1	1	9	100	8	1	6
9	9	1	1	1	9	100	12	1	7
9	9	1	1	1	9	100	1	4	4
9	9	1	1	1	9	100	1	5	4
9	9	1	1	1	9	100	1	6	5
9	9	1	1	1	9	100	1	8	5
9	9	1	1	1	9	100	1	12	7
9	9	1	1	1	9	100	2	2	4
9	9	1	1	1	9	100	4	2	5
9	9	1	1	1	9	100	6	2	6
9	9	1	1	1	9	100	8	2	8
9	9	1	1	1	9	100	12	2	8
9	9	1	1	1	9	100	2	4	7
9	9	1	1	1	9	100	2	6	7
9	9	1	1	1	9	100	2	8	7
9	9	1	1	1	9	100	2	12	8

Table 6-4. Number of resources used by HiQ for varying number or resources.

By fixing either the number of frequencies or the number of time slots, the performance of HiQ when there are abundant time and frequency resources, respectively, can be measured. When the total number of resources was greater than minimum resource number of four, the system performed significantly better in almost all cases because there was greater resource usage.

In all the experiments, HiQ s ability to allocate frequencies more effectively than time slots can be seen. When the number of total resources is the same, configurations with fixed frequencies always performed better than the same configurations with fixed time slots in terms of frequency collision and blocking rates. However, the tradeoff for this increased performance can be seen in the tag collisions. HiQ prefers frequencies over time slots, and only allocates more time slots when it is forced to because of high frequency collision rates.

6.2.5 Traffic Distributions

The final experiment using the square grid was a sixteen reader network. This simulates the HiQ algorithms ability to learn when it may not receive feedback from the system 100% of the time. RFID readers typically do not transmit 100% of the time, so it is important that an algorithm for the Reader Collision Problem can handle different and varying reader request rates which result in sparse to heavy traffic loads on the system. For this experiment, there is again one *Q*-server controller sixteen R-server, which in turn

control each reader. Four frequencies and four time slots are available for allocation. The results of this set of experiments are presented in Figures 6-21 to 6-23.

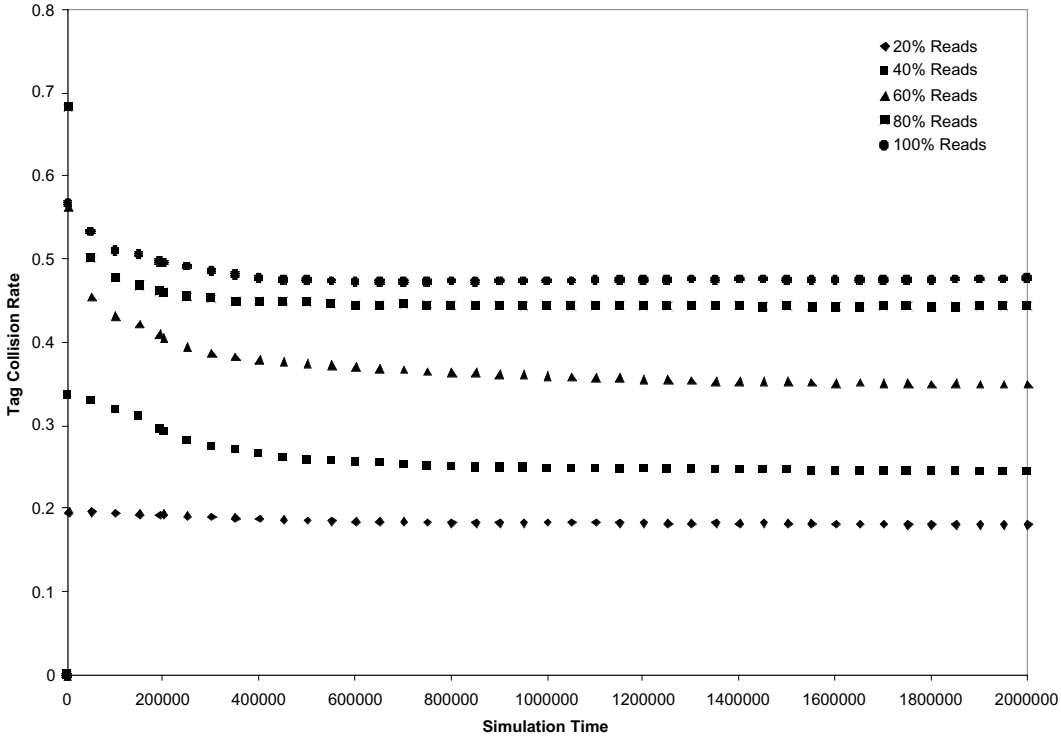

Figure 6-21. Tag collision rates for varying reader request rates.

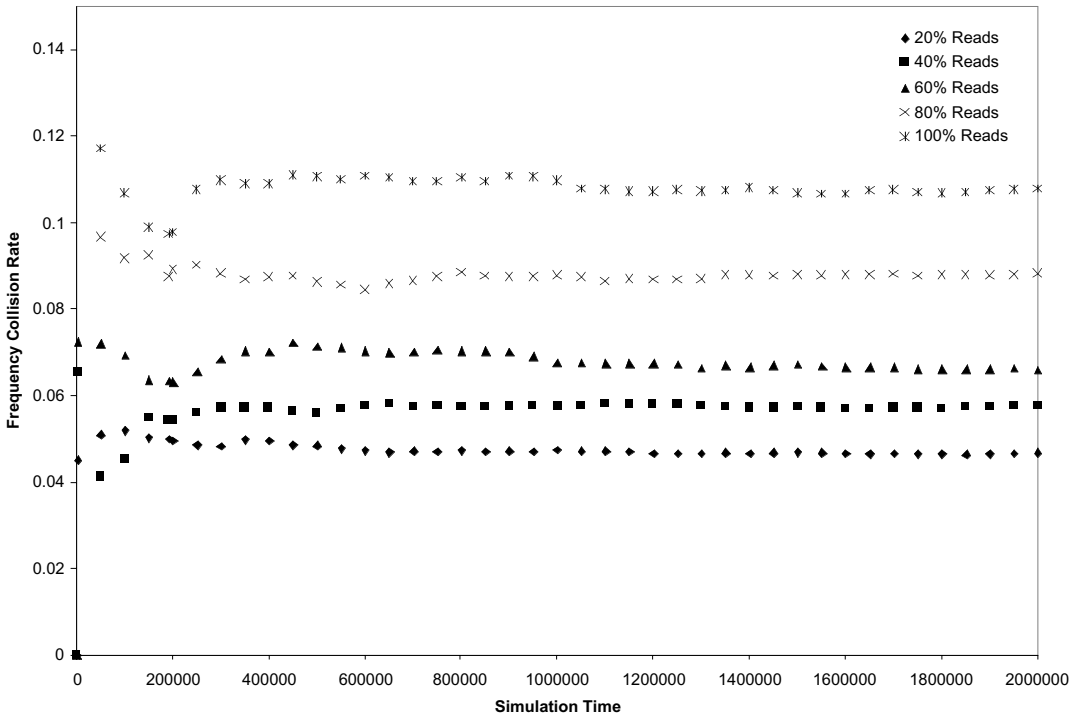

Figure 6-22. Frequency collision rates for varying reader request rates.

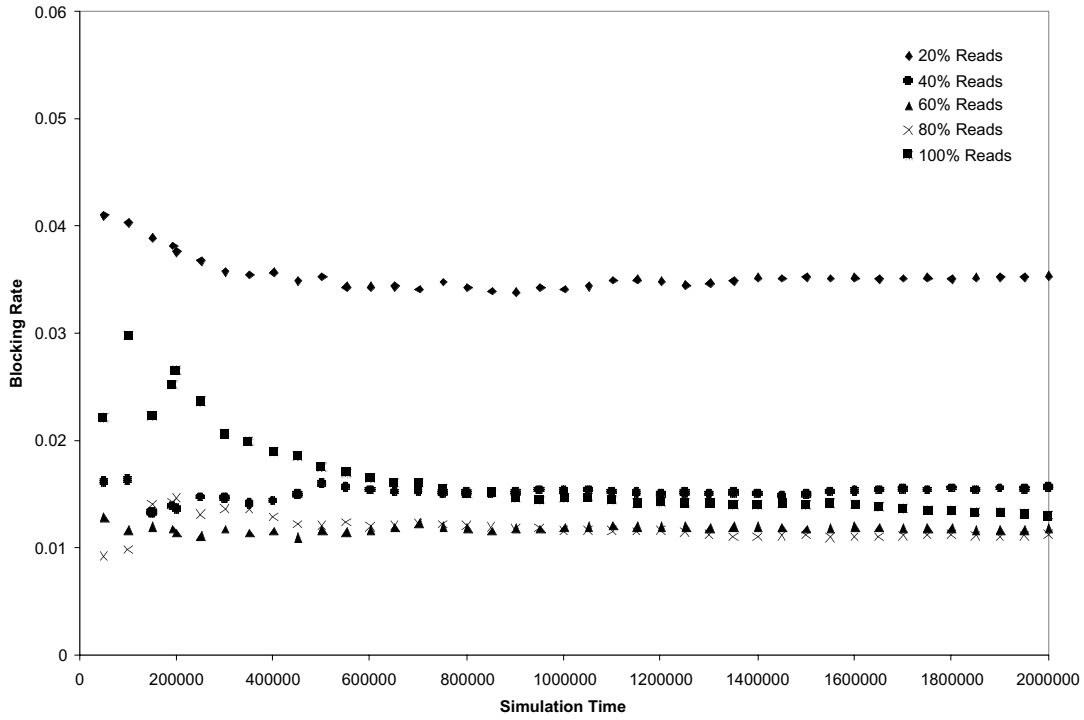

Figure 6-23. Blocking rates for varying reader request rates.

R	RS	RS/R	QS	QS T	RS/ QS	RRR	TS	Freq	Resources
16	16	1	1	1	16	100	4	4	10
16	16	1	1	1	16	80	4	4	10
16	16	1	1	1	16	60	4	4	10
16	16	1	1	1	16	40	4	4	10
16	16	1	1	1	16	20	4	4	11

Table 6-5. Number of resources used by HiQ for varying reader request rates.

From the experiments conducted with varying reader request traffic, HiQ performs better in terms of frequency collisions with lower traffic levels. However, in terms of blocking and tag collision rates, the higher traffic levels yielded better performance. This is likely due to the sparse feedback that the *Q*-servers receive during the initial learning period from networks of readers with more infrequent transmissions. Regardless of the read request rate, HiQ is able to allocate resources at levels consistently at or below 12% collision rates.

6.2.6 FAP Example

A final set of simulation was run on a large, real-world example of the Frequency Assignment Problem created by a research group at the Delft Institute of Technology and obtained from the CELAR (Centre d'Electronique de l'Armement) [18]. The specific instance used here is the eighth instance out of a set of eleven. In this example, there were 916 radio links, which we replaced with readers. There were over 5500 constraints in the system and a total of 48 frequencies available. However, each of the frequencies was broken up into smaller sets of frequency domains, so not every link could use every channel.

The HiQ configuration for solving the FAP problem was one *Q*-server with ten R-servers. There is only one time slot, as is standard for FAP. The results are shown in Figures 6-24 to 6-26.

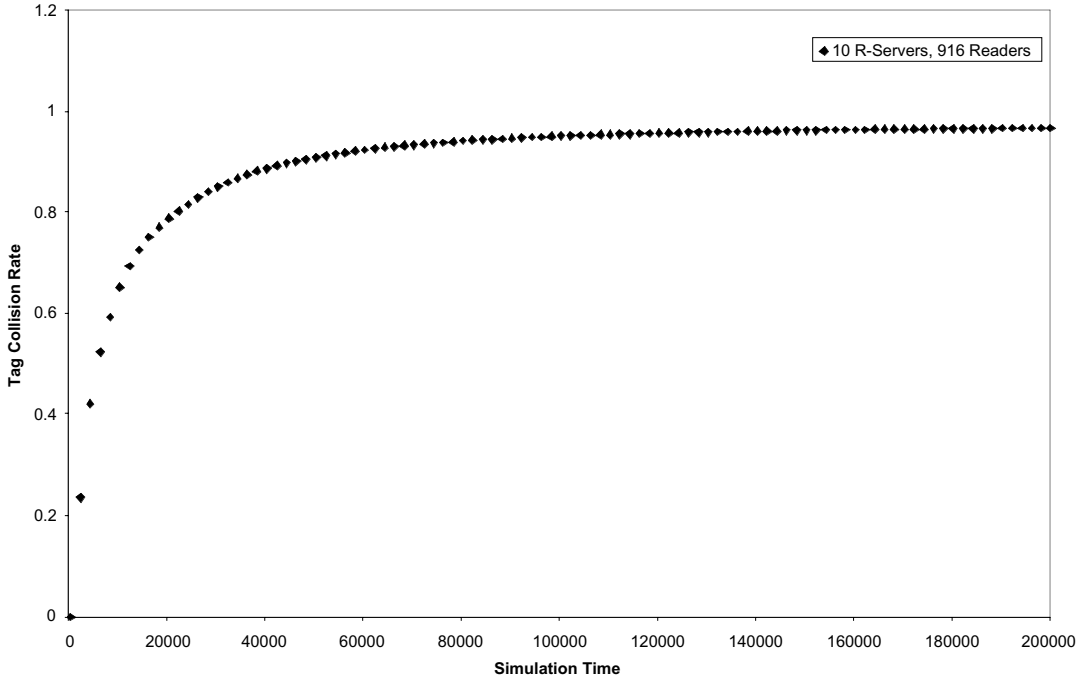

Figure 6-24. Tag collision rates, FAP example.

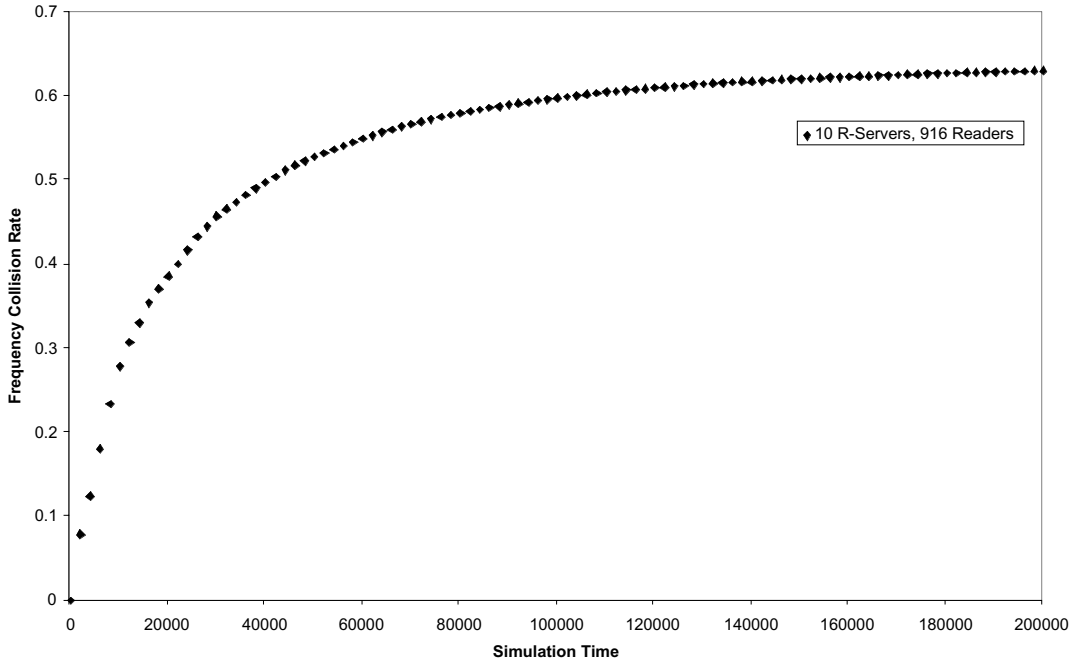

Figure 6-25. Frequency collision rates, FAP example.

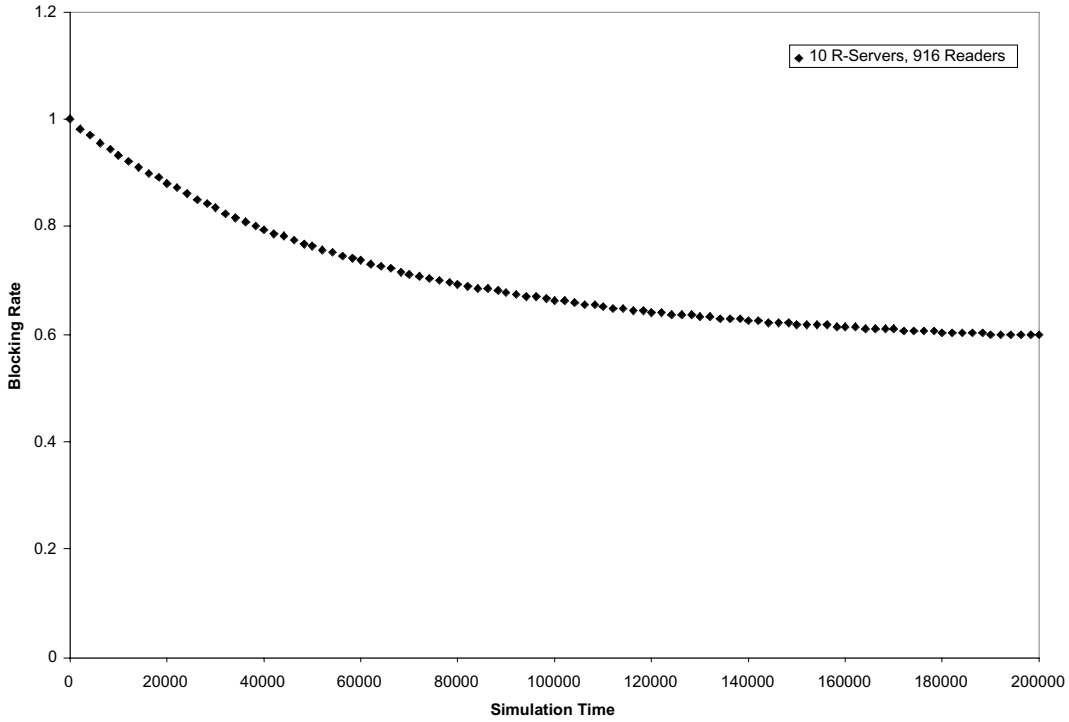

Figure 6-26. Blocking rates, FAP example.

R	RS	RS/R	QS	QS T	RS/ QS	RRR	TS	Freq	Reso urce s
916	10	91	1	1	10	100	1	48	48

Table 6-6. Number of resources used by HiQ for FAP example.

HiQ s did not perform well on the large FAP example, largely due to the lack of R-servers. Unfortunately, memory constraints due to the lookup table structure for storing *Q*-values did not allow for simulations of one *Q*-server and 916 R-servers. This would have fully tested HiQ s learning algorithm on a large data set. However, from our earlier success in the scaling experiments, we believe that if an alternate method of storing *Q*-values was used and memory requirements met, HiQ would perform well on large examples.

6.3 Results and Discussion Summary

After many rounds of simulations, the HiQ algorithm performed very well in several key areas. In almost all sets of experiments, the blocking rate of requests was extremely low. For the most part, HiQ was also able to prevent a high number of frequency collisions from occurring. There is a tradeoff between having low blocking rates and low frequency collisions rates resulting in high tag collision rates. Since the effects of a frequency collision are very similar to a request rejection (both prevent readers from communicating properly) — the two quantities are weighted closely in the cost function. The rough cost weightings of 40% for frequency collisions, 50% for rejections and only 10% for tag collisions, HiQ tends to ignore tag collisions if it means reduced frequency collisions and/or rejections.

The HiQ algorithm can effectively minimize the effects of the Reader Collision Problem when there are a moderate to large number of available frequencies. Depending on the constraints and performance criterion of a network, HiQ can be configured in many ways to address these constraints, and maximize performance.

Chapter 7

Conclusion and Future Research

In this thesis, I have presented HiQ, an online algorithm based on *Q*-learning to solve the Reader Collision Problem. HiQ combines a hierarchical structure with local control of readers to dynamically handle the potentially large and highly constrained environment of RFID systems. The algorithm is capable of effectively assigning time and frequency resources to readers without global knowledge of their constraints by measuring the performance of past assignments.

HiQ is also highly adaptable. By changing the implementation structure and configuration, the information passed up the hierarchy can be processed for optimum resource allocation. The algorithm allows for multiply-tiered *Q*-server hierarchies, providing scalability for larger networks.

HiQ allows for maximum reader request success while effectively avoiding frequency collisions and to a lesser extent, tag collisions. The algorithm is able to achieve high performance rates through a combination of effective learning and randomization techniques to avoid locally optimal solutions.

While the HiQ algorithm can perform well in systems which allow for high amounts of R-server redundancy and with an abundance of resources to allocate, there is a great deal of improvement that can still be achieved.

One of the glaring shortcomings of HiQ is its inability to minimize tag collisions. Because of the greater sub-cost weighting in the *Q*-learning formulation, HiQ prefers resource allocations that avoid rejections and frequency collisions to those that avoid tag collisions. HiQ s greater tolerance of tag collisions allows for high tag collision rates could prevent the satisfactory operation of a reader network. Raising the thresholds for allowable tag collision rates and placing more weight on the tag collision sub-cost would reduce the number of tag collisions.

Additionally, HiQ could benefit from a prolonged exploration period. In this way HiQ, could make decisions based on a longer history, avoiding convergence to a sub-optimum solution. By exploring more of the state space, the *Q*-servers have a greater likelihood of the optimum solution. Alternatively, more information could be passed along to the *Q*-servers, providing for a more detailed history rather than a longer history.

Finally, a truly distributed implementation of HiQ would be advantageous for use in RFID systems. If the computational complexity of the *Q*-learning could be reduced such that readers themselves could perform the same functions as the R-servers and *Q*-servers, the benefits could be enormous.

This thesis has provided the HiQ algorithm for solving the Reader Collision Problem, a successful implementation of the HiQ and experimental analysis of the performance of HiQ. The results of the simulations have shown that the techniques used by HiQ are effective for collision avoidance in RFID networks.

Bibliography

[1] E. H. L. Aarts, and J. H. M. Korst, Simulated Annealing and Boltzmann Machines. John Wiley & Sons, Chichester, 1989.

[2] A.G. Barto, S. J. Bradtke, and S. P. Singh, Learning to act using real-time dynamic programming. Artificial Intelligence, vol 72, pp. 81-138, 1995.

[3] R. E. Bellman, and S.E. Dreyfus, Applied Dynamic Programming. Rand Corp., 1962.

[4] M. B. Cozzens and F.S. Robert. T-colorings of Graphs and the Channel Assignment Problem. Congressus Numerantium, Vol. 35, pp. 191-208, 1982.

[5] M. Duque-Ant n, D. Kunz, and B. R ber, Channel Assignment for Cellular Radio Using Simulated Annealing. IEEE Transactions on Vehicular Technology, Vol. 42, pp. 14-21, 1993.

[6] D. W. Engels, The Reader Collision Problem. Auto-ID Center, Massachusetts Institute of Technology, 2001.

[7] D. W. Engels, S. E. Sarma, L. Putta, and D. L. Brock, The Networked Physical World System. Proceedings of the IADIS International Conference on WWW/Internet, 2002.

[8] S. Haykin, and J. Nie, A Dynamic Channel Assignment Policy through *Q*-learning. IEEE Transactions on Neural Networks, vol. 10, pp. 1443-1455, 1999.

[9] D. Kunz, Channel Assignment for Cellular Radio Using Neural Networks. IEEE Transactions on Vehicular Technology, Vol. 40, pp. 188- 193, 1991.

[10] W. K. Lai, and G. G. Coghill, Channel Assignment for a Homogenous Cellular Network with Genetic Algorithms. IEEE Transactions on Vehicular Technology, Vol. 45, pp. 91-96, 1996.

[11] R. McNab and F.W. Howell, Using Java for Discrete Event Simulation. Department of Computer Science, University of Edinburgh, 1996.

[12] R. A. Murphey, P. M. Pardalos, and M. G. C. Resende, Frequency Assignment Problems, AT&T Labs Research Technical Report: 98.16.1. Handbook of Combinatorial Optimisation, Kluwer Academic Publishers, 1999.

[13] S. Russell and P. Norvig, Artificial Intelligence, A Modern Approach. Prentice Hall, New Jersey, 1995.

[14] T. A. Scharfield, An Analysis of the Fundamental Constraints on Low Cost Passive Radio-Frequency Identification System Design. Mass. Institute of Technology Master s Thesis, June, 2001.

[15] K. N. Sivarajan, R. J. McEliece, and J. W. Ketchum, Dynamic Channel Assignment in Cellular Radio. Proc. 40th Vehicular Technology Conference, pp. 631-637, 1990.

[16] J. Waldrop, D. W. Engels, and S. E. Sarma, An Anticollision Algorithm for the Reader Collision Problem. Accepted for publication IEEE International Conference on Communications (ICC03), May 2003.

[17] J. Waldrop, D. W. Engels, and S. E. Sarma, *Colorwave:* A MAC for RFID Reader Networks. Accepted for publication IEEE Wireless Communications and Networking Conference (WCNC03), March 2003.

[18] H. P. Van Benthem, Generating Radio Link Frequency Assignment Problems Heuristically. Delft University of Technology, 1995.

[19] C. J. C. H. Watkins, Learning from Delayed Rewards. Kings College Ph.D. Thesis, May 1989.

[20] C. J. C. H. Watkins and P. Dayan, *Q-learning*, Machine Learning. Vol. 8, pp. 279-292, Kluwer Academic Publishers, Boston, 1992.

[21] M. Zhang, and T. P. Yum, Comparisons of Channel-Assignment Strategies in Cellular Mobile Telephone Systems. IEEE Transactions on Vehicular Technology, Vol. 38, pp. 211- 215, 1989.