POPULATION DYNAMICS FOR MOBILE AGENT BASED SYSTEMS

BY

BRYAN JOSHUA PROSSER

A Thesis Submitted to the Graduate Faculty of

WAKE FOREST UNIVERSITY GRADUATE SCHOOL OF ARTS AND SCIENCES

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Computer Science

May 2015

Winston-Salem, North Carolina

Approved By:

Errin W. Fulp, Ph.D., Advisor

Daniel A. Cañas, Ph.D., Chair

William H. Turkett, Ph.D.

Acknowledgments

I would like to thank Dr. Errin Fulp. Without his guidance and support for the last six years I would not have had any of the opportunities that have been presented to me.

This research would not be possible without Dr. Glenn Fink and Dr. David Mckinnon at Pacific Northwest National Laboratory. Thank you for inviting me to join your research team multiple summers in a row.

I would like to thank the rest of my committee, Dr. Daniel Cañas and William H. Turkett, as well as the rest of the computer science department for investing six years of time into my education.

Thank you to my parents and family for supporting me on this academic journey.

Table of Contents

List of Tables

List of Figures

Abstract

Bryan Joshua Prosser

Computing systems are increasingly distributed in nature and information is no longer confined to a single computer, but stored on various computers and can be under constant change. As a result traditional computing paradigms, such as the client-server communication model used for Web communications, are increasingly insufficient since they are unable to scale to these environments. New computing models will require a certain degree of autonomy that allows rapid decisions based on current, and likely local, information in order to achieve desired, long-range objectives. Mobile software agent systems have several key characteristics that are well suited for these challenging computing environments.

Many mobile agent systems rely on a population of agents roaming a network performing various tasks. The number of agents in the population is often critical for ensuring certain visitation rates throughout the network. However, it is not uncommon for the number of agents to change, due to issues within the network, such as component failure. This thesis establishes a scheme for managing the population of dynamic mobile systems by governing the death and birth of agents through the examination of the expected visitation rates of randomly wandering agents. Furthermore to avoid oscillation of the population of agents (complete death and rebirth of all agents) and visitation rates, agent queue management is analyzed. Finally this thesis performs an empirical analysis of a death and birth algorithm for agents and a queue management algorithm, that seek to prove network monitoring scalability and resilience.

Chapter 1: Introduction

As technology advances computing systems are becoming increasingly distributed. Networks of devices no longer rely on a single host device to house, process and distribute information to all of its clients. Instead the burden of processing, housing and dispersing information is distributed across a network comprised of a few to millions of devices [1, 2, 3]. As these networks grow, current schemes of communication and security can not scale to meet the needs of such a network. For instance, as a system grows it becomes increasingly difficult for human administrators to monitor said system in real time. Once a network has reached a certain size the vast number of components comprising the system may leave an administrator unable to respond adequately to an attack or failure [1]. Thus new computing models will require a certain degree of autonomy that allows a component within the system to make rapid decisions based on current, and perhaps local, information in order to maintain a secure and stable system [4, 5].

Current software technologies designed for security and network communication may also fall outside the hardware limitations of the devices that comprise distributed networks. Many of these distributed systems consist of multiple low power devices that do not have the memory and or processing power to complete their assigned tasks, while also running today's available methods of security and information distribution [1]. To meet these needs new network management and security approaches should aim to not only act autonomously, but to do so with low overhead.

1.1 The Smart Grid

One important, large scale distributed network is the smart grid [2]. The smart grid is the modernization of the traditional electrical-grid. In its simplest form, this new electrical grid seeks to embed more intelligence into devices that consume and/or monitor power. As a result, these interconnected, smart devices are no longer mere consumers of electricity, but instead are able to convey information about energy usage and costs. It is hoped that this two-way flow of information will lead to greater efficiencies than the traditional power grid [2].

The smart grid, which could include every home/office/building with a smart meter, may form a network much larger than the Internet in its current state [6]. This network will need a scalable solution to distribute data, manage operations, and provide security [2]. Furthermore, the management technique must be lightweight, since smart meters are typically low powered devices with little memory and a small amount of processing power [7]. One lightweight approach for efficiently managing network resources is the use of autonomous software agents [2].

1.2 Software Agents

An agent is a piece of software that reacts to changes and events within its environment. Agents can either be static or mobile. Static agents are often placed at one host in order to make decisions based on statistics. A mobile agent moves about its environment, often times a network, distributing and/or collecting statistics at each individual host (node).

Software agents are often autonomous and react to there environment without administrative input. An agents behavior may also influence the behavior of other agents. For instance a mobile agent may examine a hosts CPU usage. If the CPU

usage is outside the range of a predetermined threshold, the agent may incentivize other agents it comes in contact with to visit that particular host for more detailed inspections [3].

Systems using autonomous mobile agents have been suggested to solve the robust problem of distributing data and securing distributed systems such as the Smart Grid [8, 7, 9]. Mobile agents are active, software entities that move from location to location and perform a task autonomously [10]. These agents have the unique ability to make autonomous decisions during their existence (pro-actively and reactively) and to cooperate. Systems based on mobile agents can provide scalable, robust, and flexible solutions for a wide range of problems [11]. For the remainder of this thesis, the term agent will refer to autonomous, mobile software agents.

1.3 Managing Agent Populations

As described in the earlier section, mobile agents move from location to location in the network with autonomy. As an agent moves about a network it is often important to visit each node at a certain minimum rate [12, 13, 14]. For example, assuming an agents is used to monitor security, a certain visitation rate (visits per second) may be necessary for all nodes. If a single agent is unable to provide the desired visitation rate at each node, for example due to the complexity of the task and/or the size of the network, it is common to use multiple agents (a population of agents) [4, 15, 16].

Increasing the population of agents can increase the visitation rate, but it may also introduce problems. If an agent population is too large, the system will suffer, due to queuing and processing delays within the network. This is especially true when locations in the graph have different transfer times (time to travel from one host to another in a network). For instance consider using the Internet for transmitting agents and associated data throughout a smart grid. Since it is typical to encounter

several very different transmission speeds (data transmission rates) within the Internet, varying agent population sizes may be required to achieve an equal visitation rate for every node. [17]. Furthermore, since agents move autonomously, they have the ability to move between regions and as a result this may lead to an imbalance of population densities within different regions of the graph.

Increased agent population sizes to raise visitation rates may unfortunately cause congestion throughout a network. It may take time for agents to complete their task at each host. In addition, hosts with limited computing power may only support the serial processing. Therefore these situations may cause agents to build up in queues. While queued, agents are not visiting other hosts and thus not increasing the visitation rate. To insure an increased number of agents performs adequately, congestion and queue management is necessary.

To keep an equal visitation rate and to ensure that agents complete their designed task at regular intervals at all hosts, it is imperative to actively manage the agent population. A static agent population will not suffice, rather a dynamic agent population management strategy is needed to increase and decrease the agent population based on system performance [4, 15, 16]. Furthermore, population management must be distributed in design due to the scale and diversity of the network.

1.4 Thesis Contributions

This thesis aims to design and analyze algorithms to manage agent populations that are proven through a theoretical view and through empirical results. Specific contributions of this thesis includes the following.

- Develop agent birth and death processes based on theoretical visitation rates.
- Manage agent queues based on local agent measurements.

- Manage agent departures (next node visited) based on local agent measurements.
- Provide empirical results and analysis for the combination of the above contributions to manage agent populations.
- Provide empirical results and analysis for various parameter settings when combining the above contributions.
- Empirically test the scalability of the above contributions.

Chapter 2: Introduction to the Digital Ants Framework

Population dynamics can be applied to a variety of autonomous mobile agent based systems [4], however the research and solutions in this thesis were derived while working with simulators to emulate the Digital Ant Framework (DAF) being developed between Pacific Northwest National Laboratories (PNNL) and Wake Forest University. DAF consists of a mobile agents that are modeled after actual ants [1, 18]. In nature, ant drones are members of a colony and are assigned particular tasks to complete. One of the most important tasks for a colonies' survival is that of foraging in surrounding areas for both food and threats to the colony itself. When a drone finds a location of interest to the colony the drone leaves pheromones that other agents may follow to further assess what the first drone has found.

In the DAF, agents are designed to mimic a drone's behavior by wandering throughout a network. Once at at a host, an agent performs a task and reports any suspected security issues. In the interest of keeping agents lightweight, each agent is assigned to search for a specific anomaly such as high CPU utilization and memory usage. If an agent comes across a host in the network that looks suspect, the agent then leaves digital pheromone to attract other agents to the same host [19]. As more agents assess the host machine a signature can be derived identifying the cause of the anomalies. This type of wandering movement combined with stigmergic communication can lead to uneven distribution of agents in the system.

2.1 Digital Ants Framework Structure

Population Dynamics must fit the model of the agent based system that they are being applied to; thus, it is important to describe the structure of the Digital Ant Framework (DAF) further. The DAF consists of a hierarchy of agents that forms a chain of command similar to that found in an actual ant colony. At the top of the hierarchy is the Supervisor, which can be associated with that of a system administrator. Underneath the Supervisor are the Sergeants, Sentinels and mobile agents (also referred to as Sensors) [1].

The Sergeant is responsible for large swaths of the network called enclaves. The sergeant's job is to enforce policy handed down from the supervisor and assign Sentinels throughout the system. It is also given the task of communicating between the Supervisor and the Sentinels.

Sentinels are designed to manage single hosts or similar hosts, in the DAF each host is typically given a Sentinel. Sentinels provide locations for agents to reside and perform their designated task. The Sentinels also keep track of the number of agents within a subsection of an enclave. It is thus delegated to the Sentinels to create and kill agents, and therefore collectively to control the size of the agent population.

At the lowest level of the hierarchy are Sensors (digital ants), that roam a network completing their assigned task at each host. Upon completion of their task, agents report findings to Sentinels then move to another host.

2.2 Network Structure Effect on Agent Populations

Within the DAF agents have the ability to pass between enclaves. It is important to note that enclaves can be thought of as subsections of a much larger network. As mentioned in Chapter 1, each individual enclave may have different network speeds;

thus to have the same visitation rate at each host, the slower network may need more agents than the faster enclave.

Agents may also travel between enclaves; therefore, it is possible for agents to travel from fast enclaves to slower enclaves. When this occurs an agent may take a considerable amount of time to travel back to the faster enclave. In this scenario the slower enclave may have too many agents and the faster enclave may have too few. It is then necessary to develop a mechanism to allow Sentinels to monitor and manage the agent population.

2.3 Digital Pheromone Effect on Agent Populations

As mentioned in the opening paragraph of this chapter, when a mobile agent finds an anomaly it will transition from wandering to pheromone deposit mode. In deposit mode, the agent wanders away from the suspect host leaving a trail of digital pheromone pointing back towards the host. If another wandering agent comes across this pheromone trail, there is a likelihood (proportional to the pheromone strength) that the agent will follow the trail. If this agent detects an anomaly as well it well then leave a trail of pheromone back to the suspect host. This reaction to the suspect host will continue to take place creating a *pheromone map* around the suspect host and attracting agents that happens to wander into it [19]. This is depicted in Figure 2.1, where higher amounts of pheromone are located near the suspect host at the center.

As agents are attracted by the pheromone trail, the remainder of the enclave becomes void of agents and is no longer being investigated by wandering agents. In this scenario the Sentinels need the ability to create agents in the deserted sections of the enclave. However once Sentinels have reported agent's reactions to the suspect host to the Sergeant and the suspect host has been diagnosed, fixed and/or quarantined,

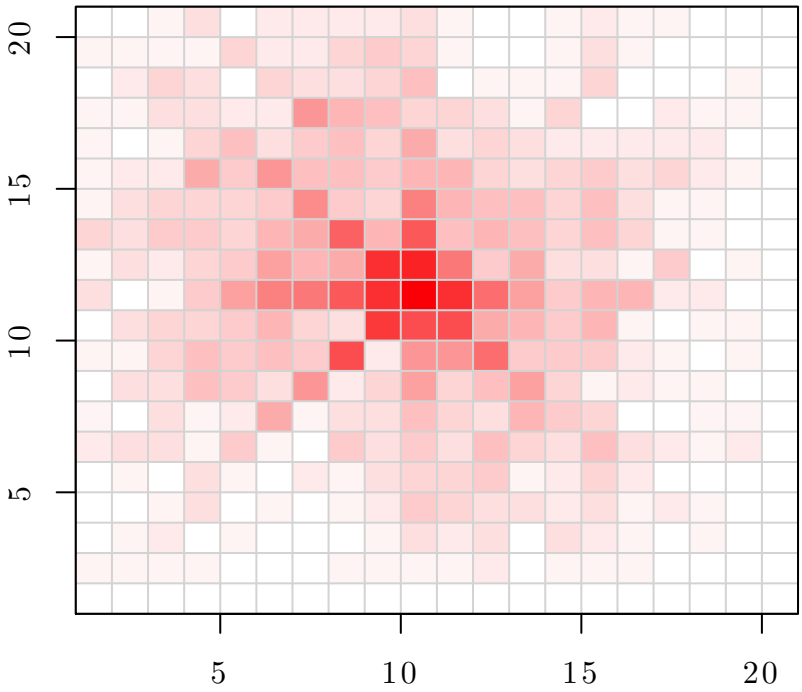

Figure 2.1: Example of a pheromone map on a 20x20 toroidal grid, where node (11, 10) is the suspect host.

the digital pheromone begins to fade and the pheromone map will reduce in size and strength. If hosts outside of the pheromone map have created more agents to compensate for agents being pulled into the pheromone map, then as the agents move away from the pheromone map the enclave agents population will have far exceeded its ideal population. The Sentinels must now be provided a way to reduce the agent population.

Definition 1. ***Ideal Population*** *The minimum number of agents needed to maintain a minimum visitation rate throughout the network.*

Agents may only deposit pheromone a predetermined number times when moving between hosts. After it is deposited, pheromone then fades at a predetermined rate. Both the number of moves made while depositing pheromone and the rate at which pheromone fades may be described using the following variables and equations:

- n is the number of hosts in a network
- p_t is the pheromone level at time t
- e is the evaporation constant

The number of moves while in deposit mode in a toroidal grid is:

$$\text{deposit moves} = \sqrt{\sqrt{n}} \tag{2.1}$$

The amount of pheromone left at a node over time:

$$p_t = \frac{p_{t-1}}{x^{t-(t-1)}} \tag{2.2}$$

For experiments, DAF agents deposit a pheromone value of 6 when visiting hosts and the evaporation constant is set at 1.25. Equation (2.1) limits the number of moves on a toroidal grid to half the grid in all directions of the suspect host.

Chapter 3: DAF Agent and Network Specifics

As mentioned in the Chapter 2, the lowest agent level of the Digital Ants Framework (DAF) consists of mobile agents roaming the network and inspecting for evidence of a security issue. Desired properties of agent movement include the quick location of a problematic host (perhaps suffering a security incident) and a consistent visitation rate for each host in the network. Overlay networks (geographies) and digital pheromone have been added to the architecture to provide these desired characteristics.

3.1 Network Topology and Geography

Mobile agent systems consist of mobile agents moving around a network. These networks can be considered as graphs, where each host is a vertex and the set of hosts that each vertex is connected to are neighbors. This graph structure, the physical interconnections of nodes, is called the network topology. For example, it has been reported that the smart grid resembles a scale free graph, while the connectivity of the Internet resembles a small-world graph [20, 21].

As depicted in Figure 3.1, it is also possible to create a network geography that is a virtual network of interconnected vertices [13]. As a result, any pair of vertices in a connected graph can be considered virtual neighbors, even if they are not directly connected. For example in Figure 3.1, node 4 (n4) in the topology only has 1 neighbor (n7), but the geography assigns 8 neighbors. Given this approach, it is possible to assign each vertex an equal number of neighbors. This creates a k-regular graph that is beneficial for population dynamics because it enables an equal probability

of visitation across all nodes in the network. In contrast, vertices in a small world or scale free graph are not guaranteed the same degree of neighbors. Without the same degree of neighbors visitation rate will not be equal, which is not desirable for agent-based security approaches.

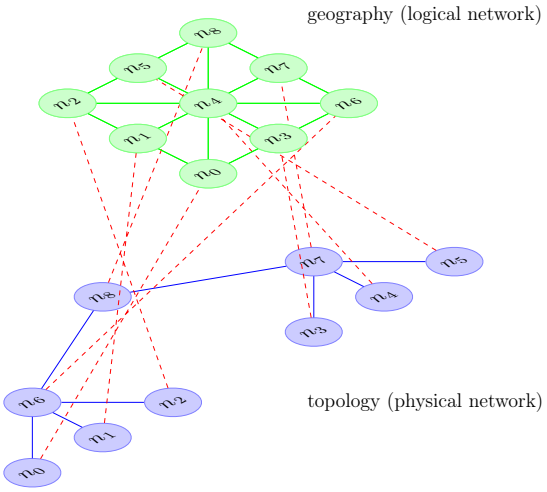

Figure 3.1: Network geography mapped to physical topology.

3.2 Agent Movement

As depicted in Figure 3.2, there are five steps associated with an agent arriving to and departing from a node. After arriving at a host, an agent is placed in a queue behind the agents that have already arrived. Agents are then serially processed, where the agent at the front of the queue is allowed to performs its assigned task. After completing its task the agent determines the host's neighbors, weights those neighbors based on movement type and pheromones, then selects a neighbor to move to. After selecting the neighbor to move to, the agent is placed in the host's transport queue

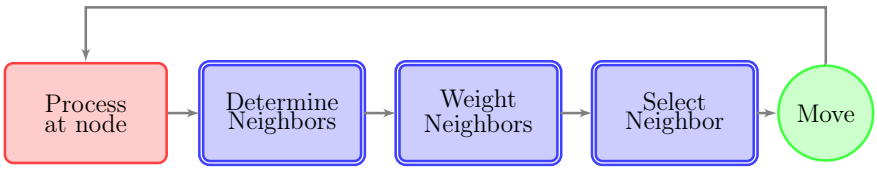

Figure 3.2: Basic subtasks associated with mobile agent processing and movement. The three subtasks associated with movement are given in double edged blocks (also in blue).

and awaits its turn to move.

3.2.1 Agent Movement Types

Better movement paradigms require fewer agents to achieve a targeted visitation rate. Fewer agents also allow for smaller and more manageable queues. However the performance of different movement schemes varies by graph types. As previously described, agent movement consists of the following three steps. First, eligible neighbors of a host are selected. Afterwards, the neighbors are weighted to influence the likelihood of moving towards a certain location. A random walk with no digital pheromone is the simplest movement scheme. When determining neighbors for a random walk, all neighbors are eligible and are uniformly randomly selected, given the same weight. Digital pheromone and heading [12] are examples of different weighting approaches designed to reduce cover time. On irregular graphs, Cooper-Abudullah is another weighting approach that can reduce the cover time by weighting neighbors based on the minimum degree of the neighbors [22].

Definition 2. ***Cover Time*** *the expected amount of time for an agent to visit each node in a graph.*

It is important to note that digital pheromone weights agent moves in the direction of a certain host. Agents may also be designed to react to pheromone in different ways.

Often the agent movement paradigm is ignored and the digital pheromone weights are applied to all available neighbors. In another paradigm, an agent may discover pheromone and continue to use its movement paradigm, and pheromone weighting is layered on top of all other selection methods. In the case of set-based heading, the hosts that the pheromone are skewed towards may not be in the set of neighbors that are possible to move to and the pheromone would essentially be ignored.

Chapter 4: Visitation Rates

The following chapter will give the definitions and analysis for the host visitation rates, that are necessary for mobile agents to monitor networks (graphs). The second section will give a brief explanation of agent population dynamic approach and goals.

4.1 Long-Term Visitation Rates

For autonomous agents to accomplish their assigned tasks (such as security of a network) it is important to ensure a non-zero minimum average visitation rate for all nodes where visitation rate is defined as:

Definition 3. ***Visitation Rate*** *is the number of agents that visit a host within a predetermined time period.*

While it is necessary to meet the minimum visitation rate, it should not be exceeded when possible. Target visitation rates may be exceeded when agent populations become too large. If the population is larger than needed, then it is likely that congestion will occur throughout the network. The agent approach should use a minimum population size while trying to provide a minimum average visitation rate. Therefore the appropriate number of agents will allow better movement by avoiding queueing delays.

As described in the introduction of DAF, it is crucial that every host in the network is visited by an agent on a regular basis. This visitation rate should remain at or above a minimum rate regardless of any network changes (network speeds, queueing delays, and/or pheromone maps). Thus it is imperative to calculate the expected number of arrivals per host. Once a minimum visitation rate has been established, actions can

be taken to correct the arrival rate by looking at the actual number of arrivals at a host during a pre-determined interval. To calculate the expected average arrival rate for a toroidal grid, the following variables are needed:

- a is the number of agents in the system
- n is the number of nodes in the grid
- k_i is the number of neighbors of node i
- m is the average agent move-time
- d is the agent density, $d = \frac{a}{n}$
- s is the sampling interval

where the average agent move-time includes the interval between the arrival at a node until the arrival at the node of the agents final destination. With all variables defined, the probability that an agent is at a particular node can be defined as:

$$P(\text{agent is at node } i) = \frac{k_i}{\sum_{j=1}^{n}(k_j)} \tag{4.1}$$

If working with toroidal grids, also known as a k-regular graphs as described in Section 3.1 the previous probability may be reduced to:

$$P(\text{agent is at node } i) = \frac{1}{n} \tag{4.2}$$

The number of moves an agent will make per sampling interval can be defined as:

$$\text{number of moves per interval} = s \times \frac{a}{m} \tag{4.3}$$

Using the probability of an agent being at a particular node in the graph (Equation 4.2) and the number of moves an agent makes in per defined interval (Equation 4.3)

the average arrival rate at any particular host can be calculated as:

$$P(\text{agent at node}) * \text{number of moves per interval} = \frac{s \times a}{m \times n} \Rightarrow \frac{s \times d}{m} \tag{4.4}$$

When determining the average visitation rate for a single agent type, it is possible to use the equations and definitions above. If more than one type of agent is used and the average visitation per type is applicable, apply the equations to each type of agent.

4.2 Visitation Rate and Agent Population Dynamics

The previous section described the long term average visitation rate for a node in the graph. However network topology, queueing delays, and the use of pheromone can affect the actual visitation rate. One method of maintaining the visitation rate throughout the network is agent population management. The general idea is that in network locations where visitation rates are low, new agents should be introduced; in contrast, in areas where the visitation rates are too high, agents should be removed.

The long term visitation rate is vital to population dynamics; however, there is no empirical data to support its use. To show that Equation 4.3 is accurate and indeed viable for use in population dynamics, simulations of agents moving about a network were performed and empirical results of average arrivals were recorded. The following chapter will describe the simulator used to obtain this empirical data and Chapter 6 will prove that the estimation provided by Equation 4.4 is accurate as well as show why population dynamics are necessary.

Chapter 5: Discrete Event Simulator

A discrete event simulator was created to test the performance of the mobile agent population management approach introduced in this thesis. The simulator models the movement of agents throughout a network over time and allows for the status of every agent and host to be recorded.

5.1 Simulator Initialization

Discrete simulators work using a global clock and an event queue to manage entity interactions. The clock keeps track of time in units applicable to the simulation taking place. Events within the simulator take place when their start times match the clock. The event queue holds all of the events and these events are held in order based on their start time. At the beginning of a simulation the clock starts at zero, and initial events are placed in the event queue.

For the Digital Ants Framework (DAF), the discrete event simulator created a multi-directional connected graph to model the desired network. Afterwards, the initial agent population was randomly placed at vertices throughout the graph. The initial events are the agent arrival times at each of their respective nodes. To ensure that population oscillations do not take place in the first interval of the population dynamic algorithm, the initial start times are staggered.

Definition 4. ***Population Oscillations*** *are large changes in the population size due to sharp changes in visitation rate every interval.*

5.2 Simulator Operation

The first population dynamic event is initially scheduled based on the interval time supplied. Thus, the event queue will contain both agent events and population events. The event at the front of the queue will have the earliest time of all events and will be removed from the queue. The clock will then be set to the start time of the event. In the case of the simulator, the first events that will take place are agents arrivals at their respective starting hosts. If there are no other agents at the host, the agent schedules its next event to be a processing event at its current host. This event takes place as the agent arrives; thus, the next event takes place at the current clock time. As seen in Figure 5.1, events include agent processing, agent arrival, agent departure, statistic events (record system state), and population management.

The process event emulates the amount of time it takes for an agent to perform a task at a host before being placed in the transport queue. After processing, an agent moves to the transport queue. This event would take place after processing, and thus the amount of processing time, so the clock time is added to the processing time to set the time of the agents next event. If an agent arrived at a host and another agent was already processing, the arriving agent would schedule its processing event as the clock time plus the remainder of the current agents processing time plus the nodes processing time multiplied by the number of agents in front of it. Departures to the next host and the departure queue events scheduling work in the same fashion as arrival and processing, except processing time is replaced by the remainder of the current traveling agent's transport time and the travel time of the agents in front of the arriving agent in the transport queue. Population events take place at the pre-defined interval. After one population event takes place, the next one is scheduled as the clock plus the interval period.

Event Type	Event Description
Agent Arrival	Agent arrives at a host.
Agent Process	Agent completes a task at a host.
Agent Departure	Agent selects a destination and moves there.
Population Dynamics	At selected interval Death/Birth Algorithm commences .
Record	At selected interval, graph and agent statistics are recorded.

Table 5.1: Simulator event types and descriptions.

5.3 Gathering Simulation Statistics

At a predetermined period, record events may take place that record and print to files the state of the network and the population that is used for analysis after simulations have been run. Examples of the different statistics gathered are the arrival rate at each host, number of agents in the system, age of each agent, queue lengths, number of agents created, and the number of agents destroyed between each snapshot that has taken place.

A simulation ends when a predetermined event takes place such as reaching a certain clock time or when no more events are scheduled and can be popped off of the event queue.

Chapter 6: Pheromone Use with No Population Dynamics

As described in Chapter 3.1, the network topology, queues, and use of pheromone can have a dramatic effect on the agent visitation rate. Using simulation, this chapter will demonstrate this effect on a toroidal grid, where an attacker is introduced then removed after a period of time.

6.1 Simulation Settings

Table 6.1 gives the average statistics over 500 simulations that use no population dynamics. The simulations took place on a 41×41 toroidal grid, had 10% agent density ($\frac{agents}{hosts}$), a total transport time of 0.007 being processing time plus transport time, and an expected arrival rate of 14 calculated using Equation 4.3. Transport and processing times were measured using an actual DAF implementation at PNNL.

The simulation consists of three phases that are associated with the introduction, detection, and removal of a single attacker in the network. During the first 100 seconds the system runs with no attacks. At 100 seconds an attack is introduced and subsequently detected by agents and a pheromone map is deposited. At 200 seconds the attack ceases and the system returns to a non-attack state. A 10% agent density was selected as this is the number of agents used in the Digital Ant Framework.

6.2 Expected Arrival Rate During Non-Attack State

During the initial no-attack phase, the average arrival rate of each host is 13.67, which is very close to the expected 14. Therefore Equation 4.3 provides an adequate

prediction of the number of agents that each host should expect to arrive. The accompanying standard deviation of arrivals is 0.06, indicating that the arrival rate is very equal across all hosts. While there is no attack, population dynamics are not needed in a k-connected network to obtain stable and expected visitation rates given the even distribution of arrivals and the low queueing levels.

6.3 Arrival Rate During an Attack Phase

Once the attacker is introduced and detected, the arrival rates outside of the pheromone map during an attack are reduced, as shown in in Table 6.1, to a rate of 2.59. This is only 18.5% of the minimum desired arrival rate for the system. At the same time, the hosts inside of the pheromone map see an increase of 18% more arrivals.

Figure 6.4 gives an example of a single simulation. During the attack phase, arrivals in the pheromone map and at the target greatly increase while those hosts outside of the pheromone map dip towards 0 arrivals. At the same time, the target's processing queue during an attack contains an average of 130.67 agents. Therefore, 77% of agents are not moving throughout the network, but instead remain in a queue, which is illustrated in Figure 6.5. Thus, the major problem during attacks is the drop in arrival rates to hosts outside of the pheromone map. Population dynamics described in later sections aim to meet the expected arrival rate at each host throughout the system at all times, while keeping the population as close to the starting population size as possible.

Experiment Statistic	**Avg. (Std. Dev.)**
Arrival No Attack	13.67 (0.06)
Arrival During Attack Outside Pheromone	2.59 (0.34)
Arrival During Attack Inside Pheromone	16.77 (1.44)
Population Size After Attack	168 (0.0)
Population Size During Attack	168 (0.0)
Queue Size Without Attack	0.09 (0.00)
Queue Size During Attack Outside Pheromone	0.01 (0.00)
Queue Size During Attack Inside Pheromone	0.10 (0.00)
Queue Size of Target During Attack	130.67 (0.96)
Agent Age	Simulation Length

Table 6.1: Statistics for 500 runs of the simulator with no population dynamics on a 41x41 toroidal grid and 10% agent population with a transport time of 0.007 seconds.

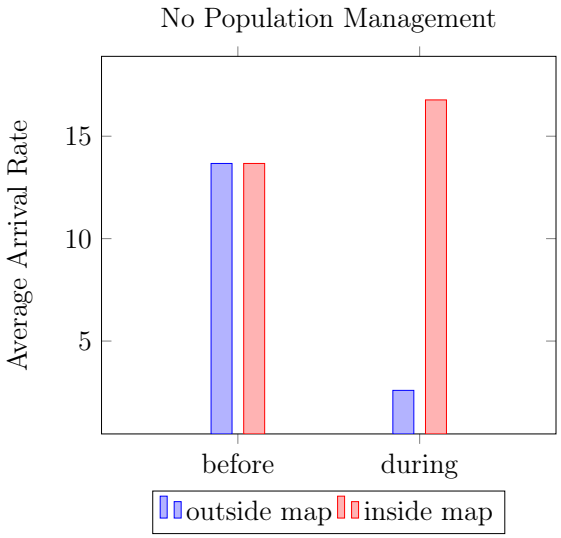

Figure 6.1: Comparison of average number of agent arrivals before and during an attack with no population dynamics.

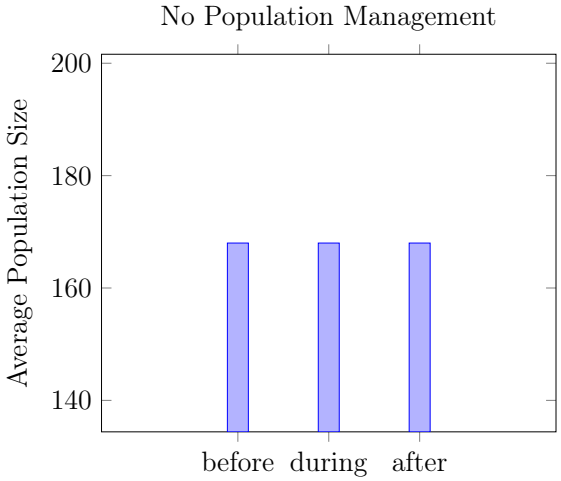

Figure 6.2: Comparison of average agent population size before and during an attack with no population dynamics.

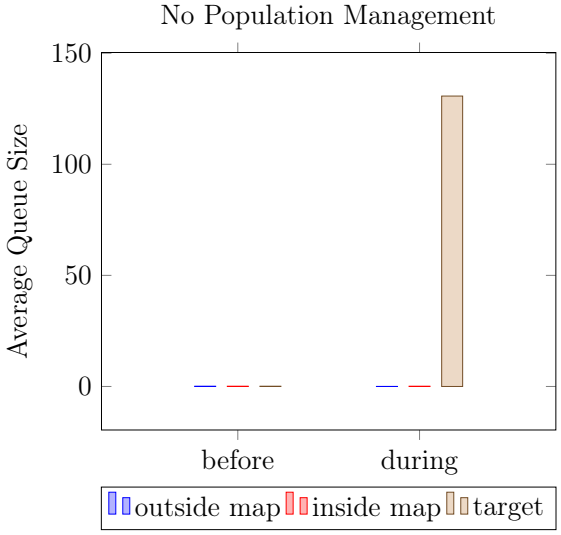

Figure 6.3: Comparison of average host queue size before and during an attack with no population dynamics.

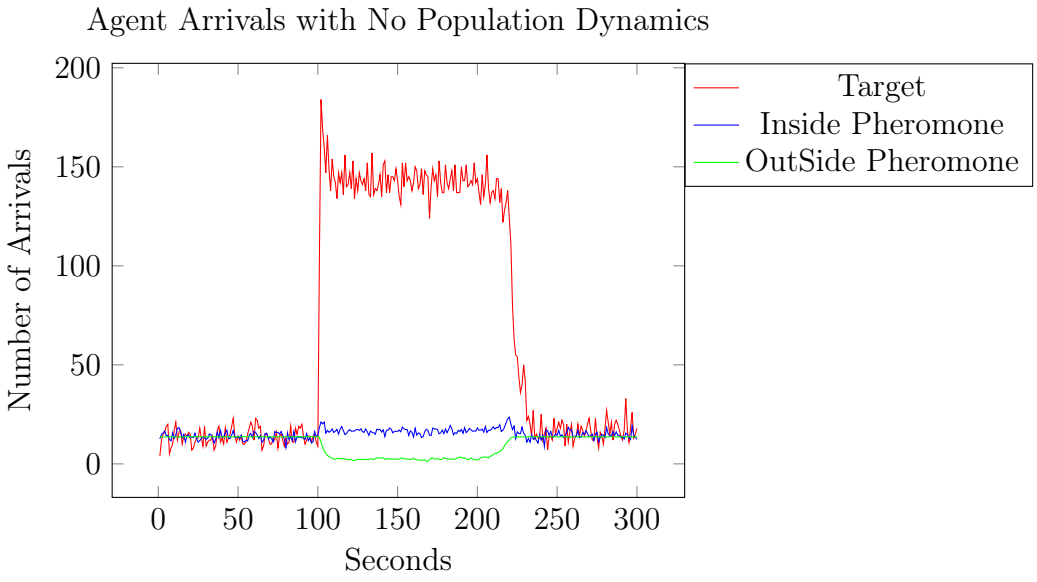

Figure 6.4: Average Arrivals for a Simulation with no Population Dynamics, using a 41x41 toroidal grid, 10% agent density and a transport time of 0.007 seconds. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

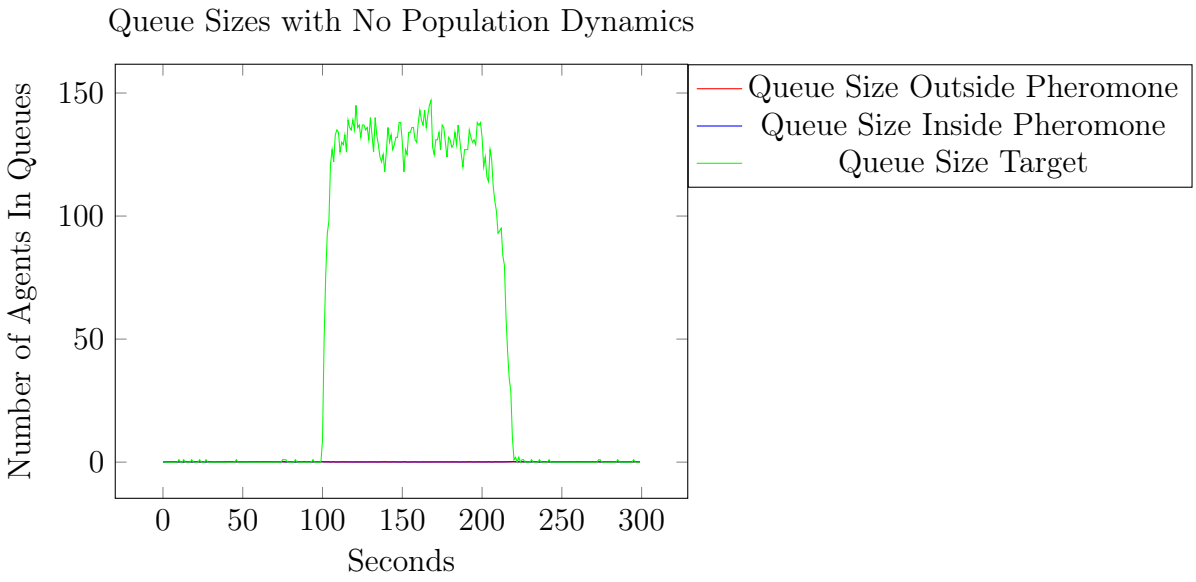

Figure 6.5: Average Queue Sizes for a Simulation with no Population Dynamics using a 41x41 toroidal grid, 10% agent density and a tranport time of 0.007 seconds. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

Chapter 7: Agent Population Dynamics

Empirical results from the previous chapter showed the drop in average arrivals at hosts outside of the deposited pheromone map during an attack. As a result, these hosts can become targets for other attackers since they are monitored less frequently. This chapter provides a solution to raise arrival rates by using the difference between actual arrival rates and estimated arrival rates to create or destroy agents, effectively raising or lowering the average arrival rate.

7.1 Birth and Death of Agents

Using the average processing speed, average transport time, and desired average minimum visitation rate, it is possible to determine the required starting population size using Equation 4.3. However, these theoretical values are based on long-term averages and do not guarantee the desired visitation rate over shorter periods of time. Therefore, it is possible that some hosts will not be visited by agents regularly enough to meet the desired minimum visitation rate determined by an administrator when measured over a smaller window of time. Also when pheromones are used, nodes outside of the pheromone map may become void of agents and the visitation rate drops at those locations as seen in Table 7.1.

These fluctuations in the visitation rate can be managed by allowing Sentinels to create agents and remove agents. Which Sentinels have the ability to manage the agent population should be randomly selected to improve system resiliency. Giving only certain Sentinels this ability could lead to compromise, since an attacker could simply target these Sentinels.

The minimum average visitation rate is used as the baseline to create or destroy

agents. If not enough agents visit a host, then that host has the ability to create an agent, and if a host has too many agents visit, it then has the ability to destroy an agent. However, if every host in the network is simultaneously given the ability to create and/or destroy agents, then population oscillations may occur. For instance, assume the agent population is above the starting value, but multiple hosts that are neighbors each have zero visits during an interval and thus each create an agent. Then there is a possibility that the arrival rate may be too high at each host after the next interval. If each of these hosts are given the ability to kill an agent, population oscillations may occur as large amounts of agents are created then immediately removed during the next interval.

To help prevent oscillations, each host is given a probability to create and destroy an agent after each interval. This probability is modeled from the p-persistent networking algorithm [23], used to manage access to a shared medium network. The probability for each host is given as follows:

1. Each host draws a number p ranging from 1 to n, where n is the number of nodes in the system.
2. If p is less than or equal to w, then the host is given the opportunity to calculate if it will create or destroy an agent. To operate correctly, $w \leq n$.

For The Digital Ant Framework, $w = a$ (the number of agents that started in the system) was used. Note that for DAF, the targeted (non-attack mode) number of agents per population is a fraction of the number of nodes [3]. With this simple algorithm it is still possible for the entire population to be restored in one interval if necessary. As a population increases, the probability that a host may create or destroy agents is reduced.

If a host has been given permission to manage population (determined using the

p-persistent approach), it will then employ the following algorithm to determine if an agent is in fact created or destroyed:

1. Calculate the percent difference between the number of actual arrivals and the expected arrivals.
2. If the percent calculated is less than a target percentage, then an agent is created. If the percent calculated is greater than the target percentage, then the process of agent destruction is started for that host.

Agent destruction consists of destroying the next agent that arrives at the host. To decrease the chance of population destabilization and oscillation, destruction should happen less than agent creation. Thus one would limit those hosts that are given permission to destroy agents to 1/3, which was determined empirically. To do this, a random number is generated at a host whose arrival rate is above the expected threshold, and if that number modulo 3 = 0, then the host is given permission to destroy an agent during the next interval, else permission to destroy is not given and the number of agents at the given host is kept the same. To help reduce the risk of agent population oscillation, population events that use this algorithm are staggered for each host throughout the network.

7.2 Birth and Death Algorithm Results

Table 7.1 gives the average statistics over 500 simulations run on a 41×41 toroidal grid, with 10% agent density, total transport time of 0.007, an expected arrival rate of 14, and a birth/death threshold of 50%. During no attack, Table 7.1 shows an arrival rate of 14.662. This is an increase over 13.67 seen in Table 5.1. This higher arrival rate signals that some hosts in the network created agents, raising the general population level and raising the average arrival rate.

The birth algorithm is implemented to increase the average agent arrival rate for hosts outside of the pheromone map during attacks. Table 7.1 shows that this goal is clearly met as those hosts have an arrival rate of 31.05 agents. This is over double the calculated expected arrival rate. It can also be seen that during the attack the average agent population reaches 22,712.75, which is 135 times the original number of agents that the system contained. Of those 22,712.75 agents, 53% are in the target host's queue. With such a high arrival rate it would be expected that the death algorithm would be activated and the population would decrease. But, average queue lengths within the pheromone map are 14, which suggest that most agents that are created outside the pheromone map are pulled into the pheromone map and are thus not subject to being destroyed. Also, with so many agents in the target queue, the agent population does not have a chance to decrease.

An example of the population during one of the 500 runs is shown in Figure 7.5. Here it can be seen that the population quickly and continually grows during the attack. So many agents are created that even after the attack, the system struggles to return to the non-attack population levels and average arrivals. A large agent population accompanied with a large arrival standard deviation (25) outside the pheromone map support the conclusion that agents are continually being attracted into the pheromone map once they have been created. The oscillations of arrival rates seen in Figure 7.4 display this high standard deviation throughout the attack. Oscillations in arrival rate continue even after the attack has finished. The system can not recover from the increase of agents, even as they begin to spread out across the network. It can be concluded that the algorithm can not handle the substantial increase in agents from the starting population size.

Experiment Statistic	**Avg. (Std. Dev.)**
Arrival No Attack	14.66 (0.30)
Arrival During Attack Outside Pheromone	31.05 (25.24)
Arrival During Attack Inside Pheromone	39.23 (13.30)
Population Size After Attack	24610.60 (3.99)
Population Size During Attack	22712.75 (6224.47)
Queue Size Without Attack	0.11 (0.00)
Queue Size During Attack Outside Pheromone	0.29 (0.00)
Queue Size During Attack Inside Pheromone	14.85 (0.97)
Queue Size of Target During Attack	12056.54 (243.56)
Agent Age	97.06

Table 7.1: Statistics for 500 runs of the simulator with Birth Death using a 41x41 toroidal grid, 10% agent population, 0.007 total transport time, expected arrival rate of 14 and a birth/death threshold of 50%.

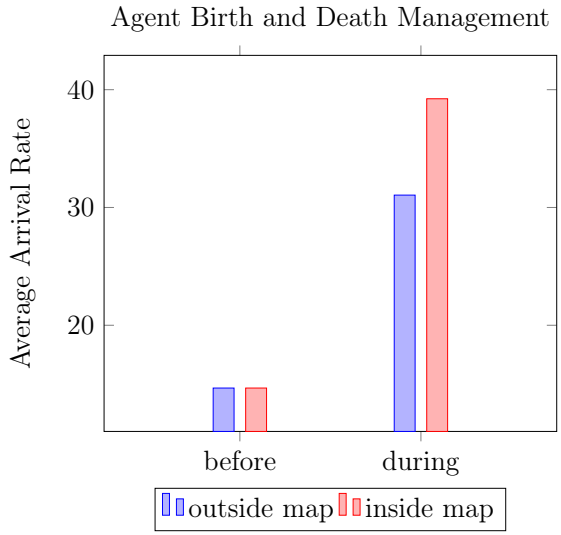

Figure 7.1: Comparison of average number of agent arrivals before and during an attack with birth and death management.

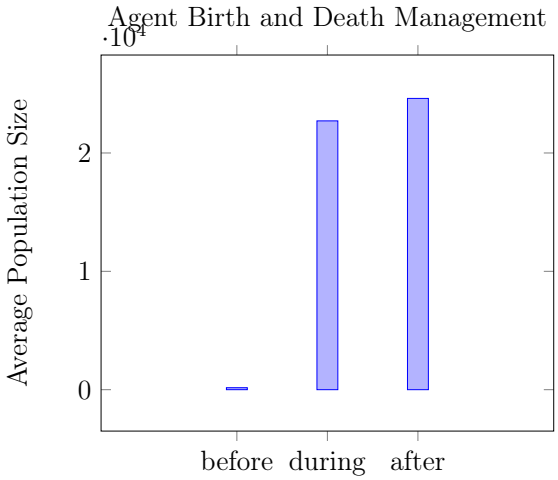

Figure 7.2: Comparison of average agent population size before and during an attack with birth and death management.

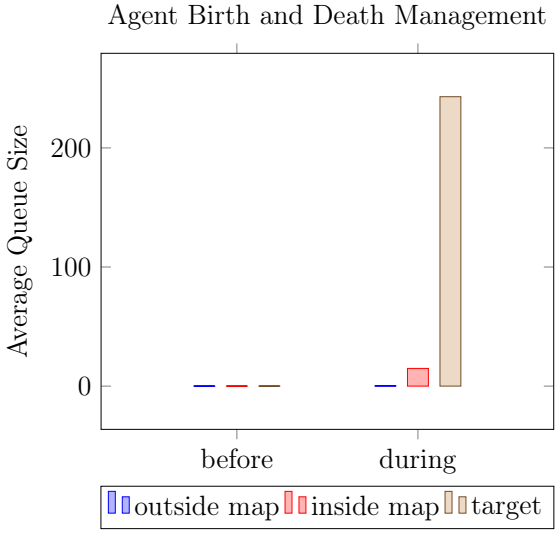

Figure 7.3: Comparison of average host queue size during and after an attack with birth and death management.

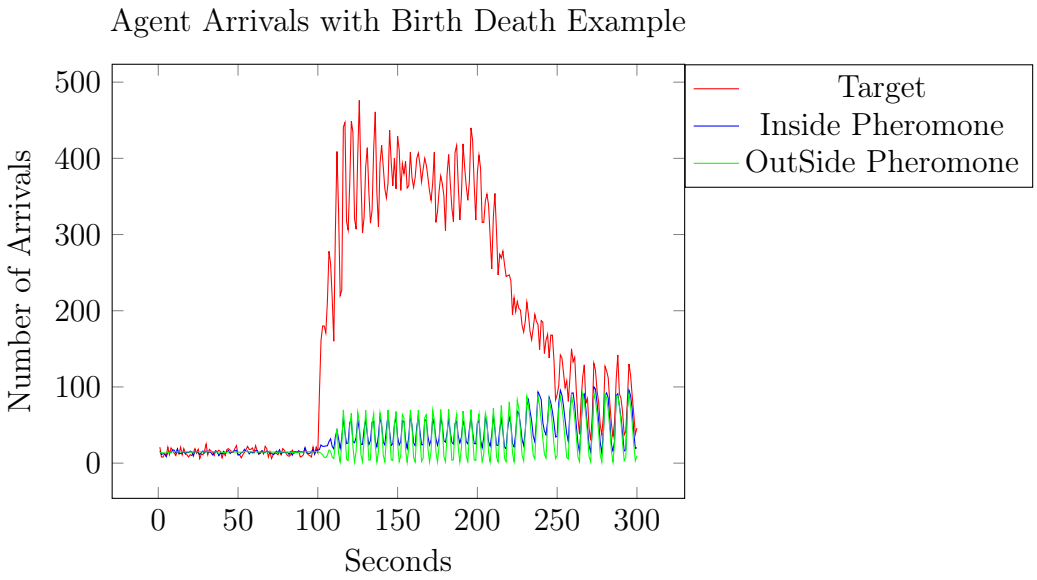

Figure 7.4: Average Arrivals for a Simulation with Birth Death using a 41x41 toroidal grid, 10% agent density, 0.007 total transport time, expected arrival rate of 14, birth/death threshold 50%. The system runs normally for 100 seconds. Then an attack take place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

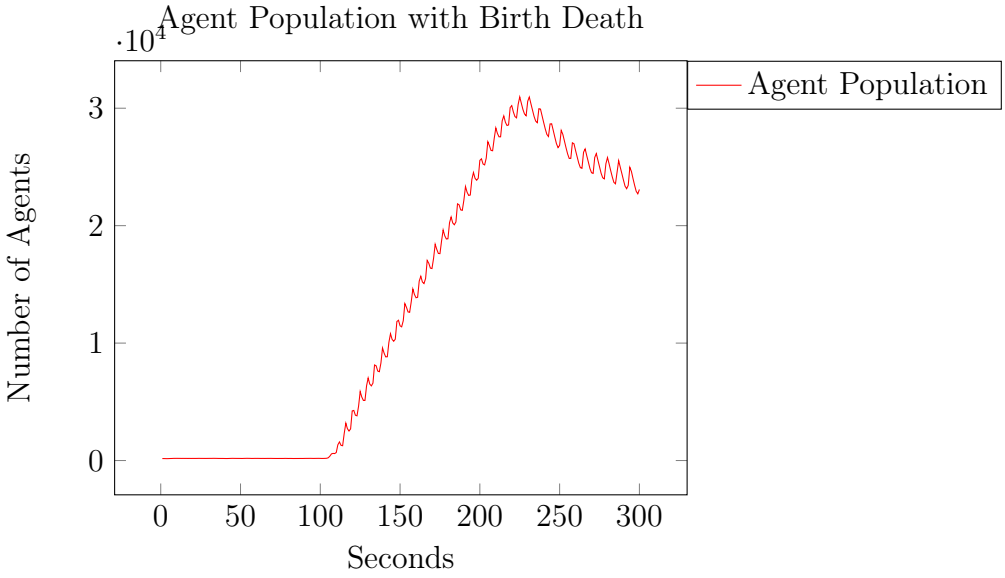

Figure 7.5: Average Population Size for a Simulation with Birth Death using a 41x41 toroidal grid, 10% agent density, 0.007 total transport time, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack take place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

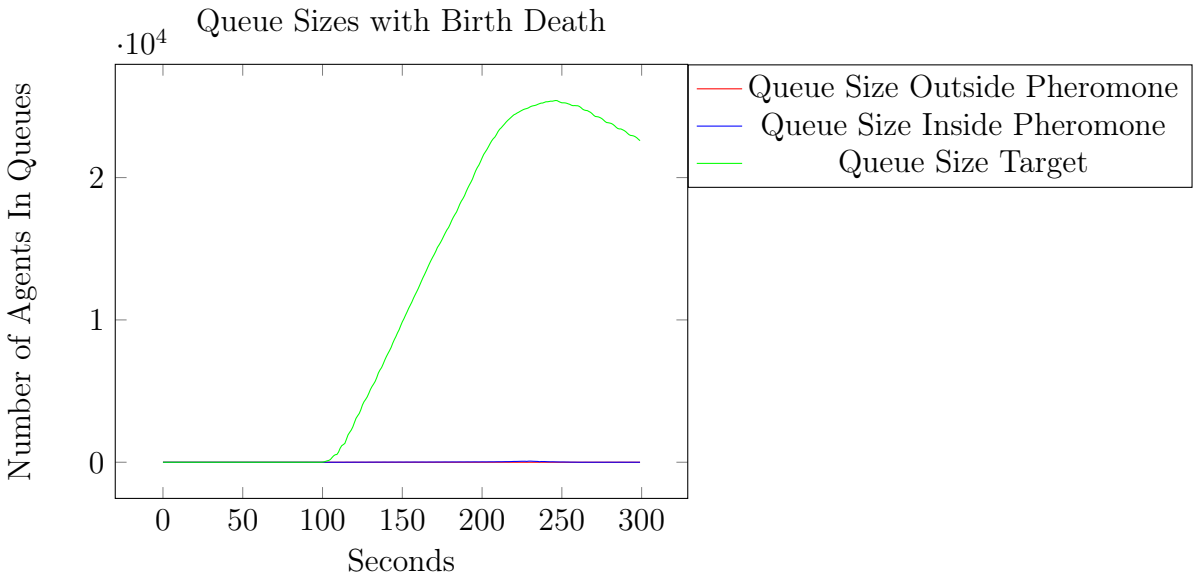

Figure 7.6: Average Queue Sizes for a Simulation with Birth Death using a 41x41 toroidal grid, 10% agent density, 0.007 total transport time, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack take place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

Chapter 8: Queuing Algorithm

As discovered in Chapter 7, an algorithm that creates and kills agents is not sufficient for maintaining a stable minimum population of agents. Many of the poor statistics seen in Table 7.1 derive from agents waiting in long queues. To help prevent large queues at the target and within the pheromone map, a simple but effective algorithmic approach was implemented to limit queue sizes and direct agents to less congested neighbors.

8.1 Limiting Queues

As demonstrated in the previous simulations, agent queues can become exceedingly large. One approach for managing long queues is to implement a *tail drop* policy (as done in computer networks), where agents arriving to a full queue are simply dropped [23]. The maximum queue length can be set by network limitations or decided upon by a system administrator. The next section describes an algorithm to limit how often this may occur.

8.2 Estimating Congestion of Neighbors

A queue management algorithm was developed based on estimating the congestion of queues at neighboring hosts. However, since many agent based systems must be designed with hardware and network limitations, it is desirable to implement this estimation without requiring each host to continually broadcast queue lengths to its neighbors.

The average number of agents sent by a host is used to estimate congestion of a neighbor. Each host has a processing time, and a host can not send agents faster than

this processing time. This processing rate will be called q and the neighbors of a host should collectively never send agents at a rate more than q. This approach is based on a $M/M/1$ queuing analysis, which can be used to estimate delays. $M/M/1$ queuing states that the queue size approaches infinity when the total arrival rate equals the total processing rate [23]. Given k neighbors and no neighbor sending agents more often than any other, each neighbor should send at a rate no higher than $\frac{q}{k}$, where q can be estimated as 1/(average processing + transport times). The goal is to then limit the number agents sent to each neighbor to this predicted rate.

Implementing this algorithm requires each host to record the time an agent was sent, up to the past ten agents per neighbor. The sending rate is the number of agents sent to a particular neighbor divided by the latest departure from any queue minus the earliest departure of the particular neighbor queue that the agent was sent to. If this rate is above the expected value $\frac{q}{k}$ by a predetermined percentage than a new neighbor is selected. The next neighbor is selected from the same set of possible neighbors that weighting and movement type had returned before selection, minus the neighbor that is known to have too high a send rate.

When an agent goes through selecting its next neighbor, it goes through the original process of movement type and weighting schemes. After a neighbor has been selected, the above calculation takes place for the new neighbor. If the rates for each neighbor in the original set of moves is too high, those neighbors that were excluded by the weighting and movement models are than considered. If all neighbors have been disqualified than a random neighbor is selected for the agent to travel to.

8.3 Queue Management Result

With queue management implemented, simulations were run on 41×41 toroidal grids, with one type of agent at 10% density, which is 168 agents. The average expected

Experiment Statistic	**Avg. (Std. Dev.)**
Arrival No Attack	14.65 (0.30)
Arrival During Attack Outside Pheromone	17.35 (1.18)
Arrival During Attack Inside Pheromone	65.36 (4.32)
Population Size After Attack	180.94 (3.99)
Population Size During Attack	471.59 (31.04)
Queue Size Without Attack	0.10 (0.0)
Queue Size During Attack Outside Pheromone	0.10 (0.0)
Queue Size During Attack Inside Pheromone	1.8 (0.31)
Queue Size of Target During Attack	5.79 (2.35)
Agent Age	64.19

Table 8.1: Statistics for 500 runs of the simulator with Death/Birth and Queue Management using a 41:41 toroidal grid, 10% agent population, expected arrival rate of 14 and a birth/death threshold of 50%.

arrival was set at 14 which gives an interval time of about one second. The maximum queue length was set to 300, with a 50% threshold for agent creation and death. Processing time was set at 0.006 time units and transport time was set at 0.001 units. The agents were set to roam about the grid for 100 seconds. At 100 seconds a random host was attacked. Once an agent finds the attacked machine it would result in a pheromone field being created. Agents were enabled to lay down pheromone in half the square root of the total number of host steps (which is approximately 20 steps for this grid). The target host was attacked for 100 seconds. At 200 seconds the attack stopped and the host was no longer suspect, stopping the spread of pheromone and allowing the system to return to a normal state. At these settings the simulation was run 500 times and the average results were recorded.

8.3.1 Effects on Queues

From previously examined runs without queue management it can be seen that the queues during the non-attack phase maintain a reasonable length, as seen in Table 7.1. However once pheromone has been dispersed queues begin to build and without

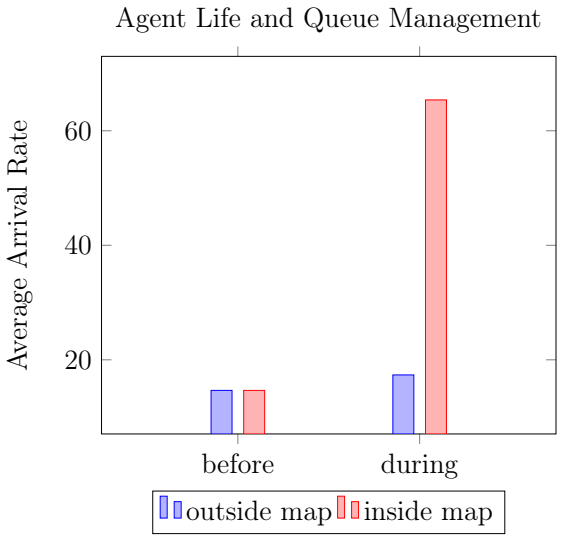

Figure 8.1: Comparison of average number of agent arrivals before and during an attack with queue and birth/death management.

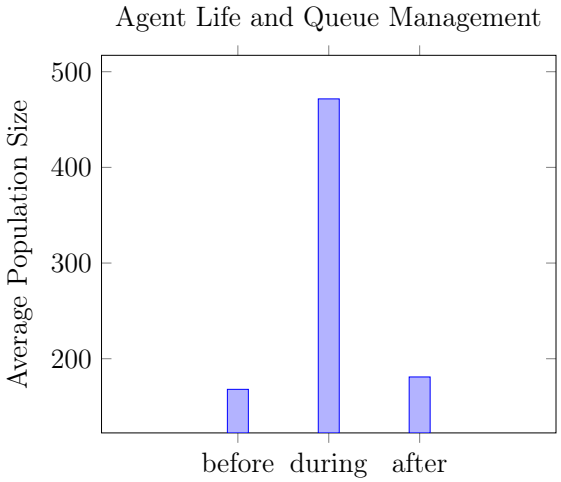

Figure 8.2: Comparison of average agent population size before and during an attack with queue and birth/death management.

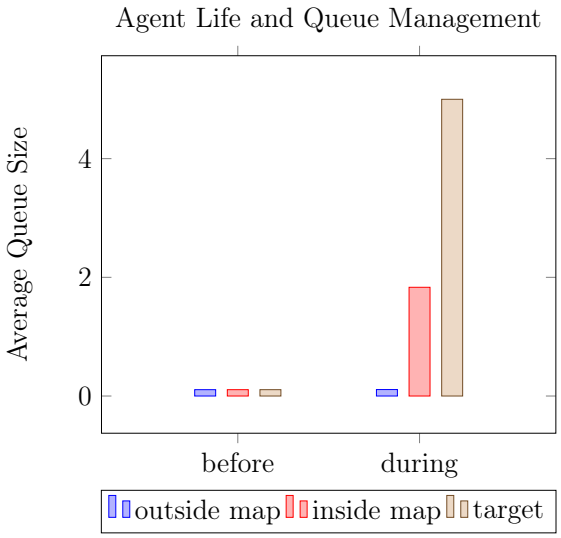

Figure 8.3: Comparison of average host queue size during and after an attack with queue and birth/death management.

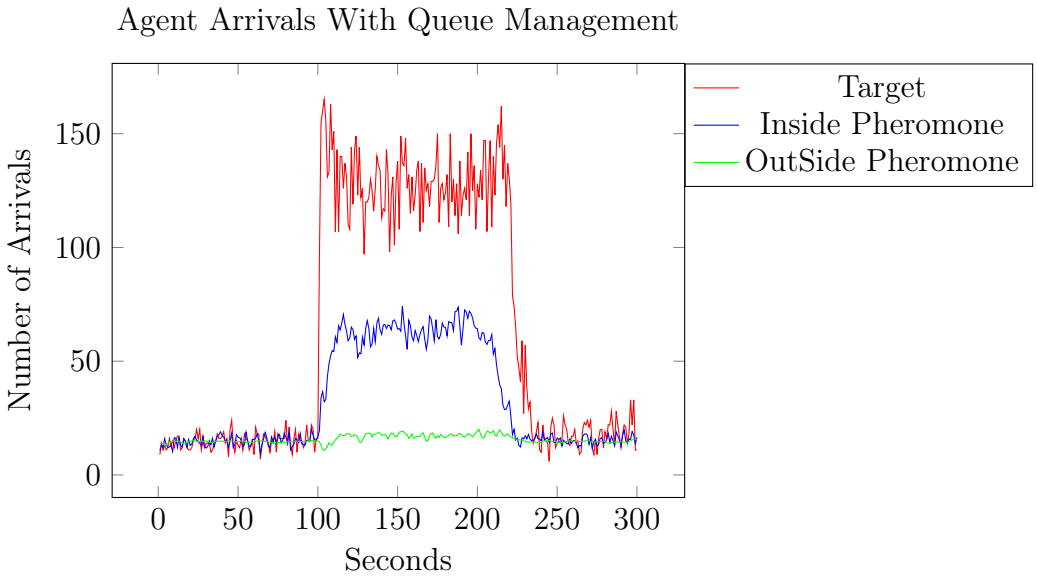

Figure 8.4: Average Arrivals for a Simulation using birth death and queue management using a 41x41 toroidal grid, 10% agent density, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

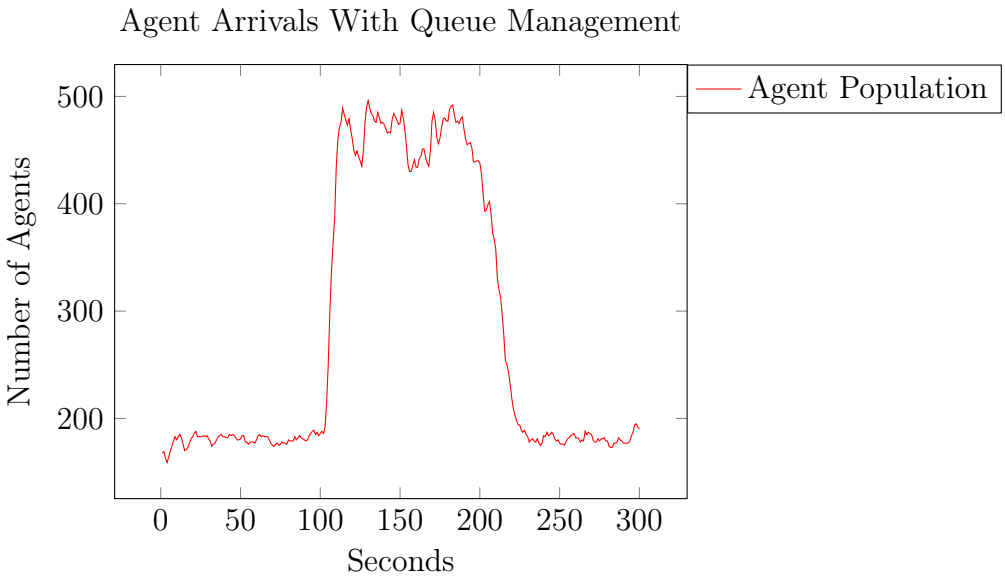

Figure 8.5: Average Population Size for a Simulation using birth death and queue management using a 41x41 toroidal grid, 10% agent density, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

queue management grow to be excessively long. With queue management the queue lengths become shorter. Table 8.1 depicts the targets queue having an average of 5.7 agents in it. Looking at an example simulation in Figure 8.6 the queue length does not even reach 80 agents at its peek, which is very shy of the 300 cap limit and .006% of the queue size of the target in Table 7.1.

Figure 8.6 shows an initial spike at the target node when the attack starts, and is common amongst all runs. It is caused by all eight hosts neighboring the target sending all their arriving agents to the target due to pheromone. All of these agents arrive simultaneously until each of their send rate thresholds are met. Once this threshold is passed, agents are diverted to neighbors of the target. This process continues until agents are sent backwards through the pheromone field until the percentage of agents sent to the target drops below the threshold and agents are again sent to the target.

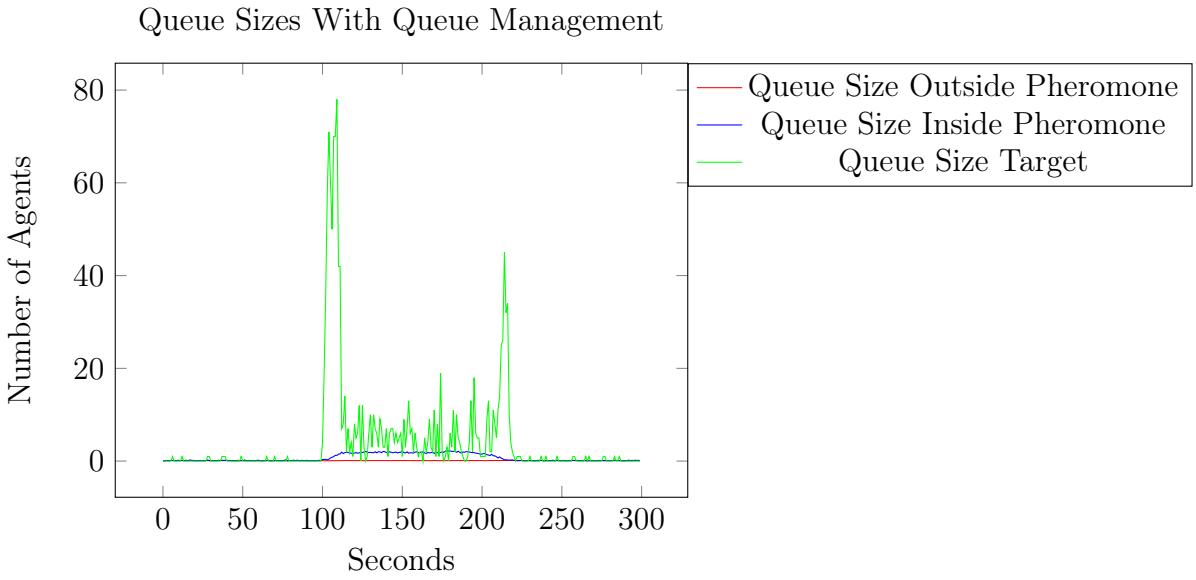

Figure 8.6: Average Queue Sizes for a Simulation using birth death and queue management using a 41x41 toroidal grid, 10% agent density, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack.

Agents follow pheromone towards the target and once they reach the target move away from the target as long as they are dropping pheromone. Many agents that reach the edge of the pheromone field after they stop depositing pheromone again start following the pheromone trail back towards the target. This is what causes the queue length to stay pretty stable if not grow a little as more agents get trapped in the pheromone field during the attack. After the attack it can be seen that all queues quickly return to normal length around 0.

8.3.2 Effects on Population Size

It should also be noted that queue management has a strong affect on the agent population size. Table 8.1 shows that the average population size during an attack is 471 agents. This is approximately 2% of the agent population of an attack when

queue management is not used. Queueing management does not allow hosts to be sent over a defined rate. As agents are pulled into the pheromone field, the rate at which agents are sent between hosts inside the pheromone map hits the threshold and agents can no longer move towards the target. This congestion of sorts moves outwards and does not allow new agents to enter the pheromone map after it has become saturated. While agents are being kept out of the pheromone field, the arrival rate stays high enough outside the pheromone field that more agents are not created.

8.3.3 Effects on Agent Visitation

There are also effects on agent visitation rates. With queue management, during attacks, agents are kept outside the pheromone field as explained above. With more agents kept outside of the pheromone map the chance that areas outside the pheromone map may becoming void of agents and visitations is rare. As queue management keeps more agents outside of the pheromone map, hosts outside of the pheromone map are able to meet the expected arrival rate without creating an excess of agents. Obtaining the average visitation rate across the network prevents oscillations within the population The lack of oscillation can be seen in the smaller standard deviation of arrival rates throughout the attack period and the non attack period that follows. Steady arrival rates during all portions of the simulation can be seen in the example of arrival rate in Figure 8.4.

Chapter 9: Parameter Testing

Although an approach has been established for both death and birth intervals along with queue management, there are several management parameters that must be correctly set to appropriately manage the agent population, in particular the average expected arrival rate and the threshold for the difference between expected arrival and actual arrival that controls the death and birth of agents. To analyze the effect of tuning these parameters, simulations were run with the same parameters as those in the queuing experiments. However, the number of agents expected varied from 6 to 30, which changed the interval period according to Equation 4.3. At each of these expected arrival rates, the death and birth threshold varied between the range of 25% and 80%. There are two parameters not tested that may also have an effect on agent populations. One is the percent difference between the maximum send rate and actual send rate to each of a host's neighbors, used to gauge congestion. Another is maximum queue size.

To analyze how parameters affect the state of an agent population, many different statistics were examined, in particular the standard deviation of the population size during non-attack phase and attack phases, the average population size during both phases, the standard deviation and averages of arrival rates during both phases inside and outside of the pheromone field, and the average age of agents throughout the simulations.

When looking at the following graphs, it is important to keep in mind that the statistics are kept based on the record event interval, not the population interval. Therefore, the record interval for all runs was set to one second. So if all time intervals perform equally, every plot point should be the same for each respective threshold,

because no matter what the population interval is, the same arrival rate over a one second period is expected.

9.1 Arrival Rates without Attacks

Using Equation 4.3, it was determined that for a 41x41 toroidal grid with 10% agent density, with a transport time of 0.007, and a record interval of 1 second, approximately 14 agents should be seen per second. Looking at each of the plots in Figure 9.1, the average arrival rate decreases as the population event interval decreases. The smaller the interval the more often the population is corrected. As the population is corrected more often, there is a higher chance at creation and destruction of agents. However as described earlier there is a higher chance of creation than destruction of agents. This leads to higher visitation rates for lower time intervals. The idea that there are larger changes in the population is supported by the Standard deviation of arrivals when attacks are not happening. Again as seen in Figure 9.5, as the population interval becomes longer the Standard deviation of arrivals decreases.

Figure 9.1 indicates that as the threshold increases the average visitation rate decreases. As a larger disparity between actual arrival rate and expected arrival rate is required for agents to be created or destroyed less agents are created. Fewer agent creations result in fewer visitations. The interesting cases occur at the highest thresholds. It can be seen that at the 0.75 and 0.80 thresholds arrival rates decrease until an interval expects 16 arrivals and as the intervals increase so does the average arrival rate. This trend is opposite of the lower thresholds which all increase.

This trend takes place when there are not enough agents to meet the expected arrival rate at which point the disparity between actual and expected arrival rates is great enough that it passes the higher threshold causing the system to overcompensate creating multiple agents. The destruction threshold in these simulations are set at

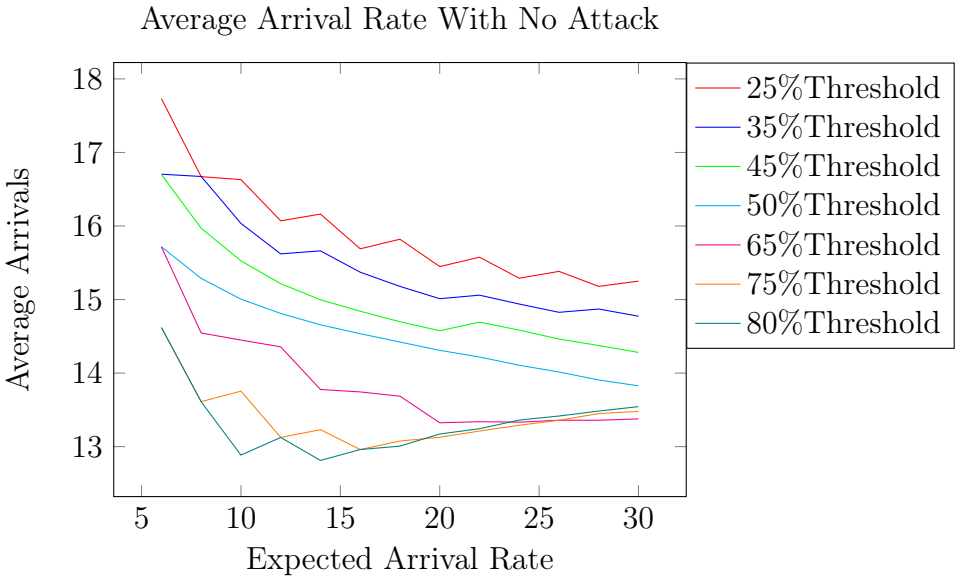

Figure 9.1: Average Arrival as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

the same level as the creation threshold, making the destruction of agents as unlikely as creating them. Thus once the agents are created they are never destroyed, creating an upward trend in average arrivals as the interval increases. This is supported by the standard deviation of agent arrivals, even with the increase of agents and arrivals, the standard deviation is close to 0. From this explanation the average arrival rate can be expected to continually increase as the expected number of arrivals increase until to many agents are created and deaths are triggered. At that point the average arrival would again begin decreasing and the standard deviation will stay small or the population will be thrown into oscillation at which point the standard deviation of arrivals will look closer to that of the 0.25 threshold.

During normal operation without an attacker one would like the average arrival rate to be around 14 agents per second. The threshold of 0.65 with an expected arrival of 14 provides the desired rate while retaining a low standard deviation of

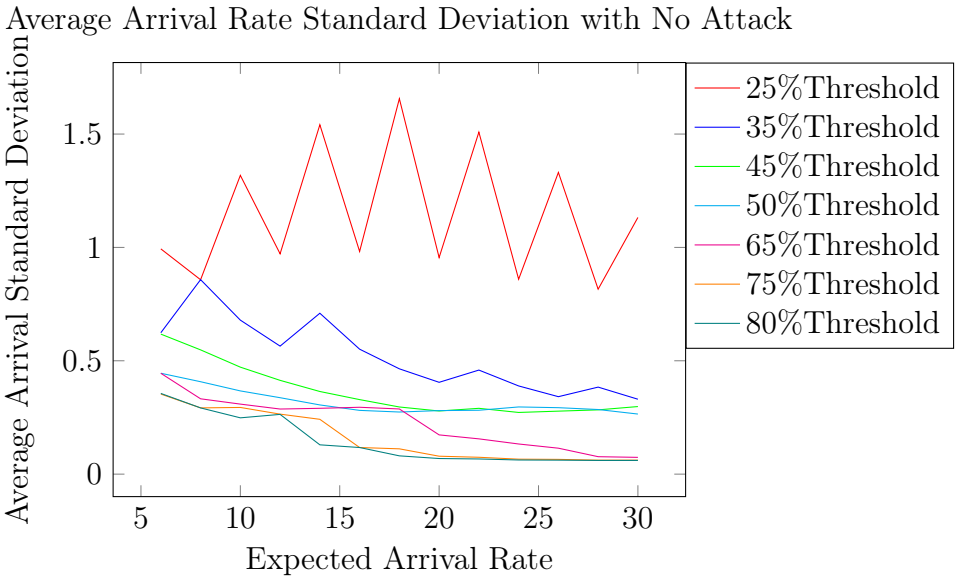

Figure 9.2: Average Arrival Rate Standard Deviation as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

approximately 0.35. However all of the settings provide a reliable average arrival rate while never exceeding an arrival standard deviation of 1.7 in the worst case.

9.2 Agent Population No Attack

Figure 9.3 shows that for most thresholds a shorter interval means an increased population. This matches the reasoning for average arrival rates with the same settings. When an interval is short, then less agents are expected and the lower thresholds are easily affected. With less time for agents to make moves and less agents than hosts, multiple hosts will not receive the number of visitations necessary to meet the number of expected arrivals during a shorter interval, thus creating a larger population to meet the average expected arrival rate.

The previously stated scenario also creates higher standard deviations as seen in Figure 9.4. With low thresholds for creation of agents comes low thresholds for the

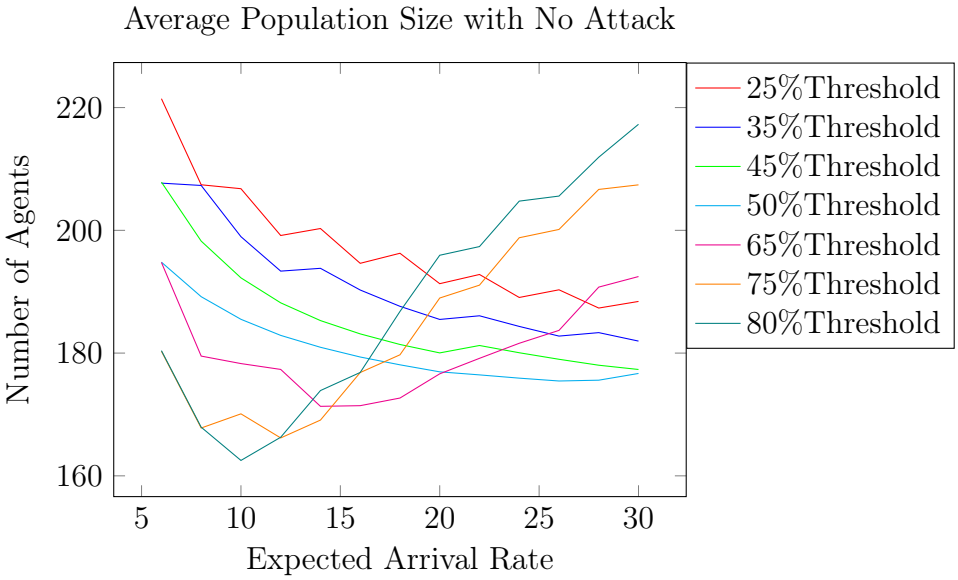

Figure 9.3: Population size as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

destruction of agents, causing oscillations within the population size. Again high thresholds combined with a high expected number of arrivals caused some interesting effects within the population. As stated before, at some point arrivals are low enough and high thresholds are met. At this time multiple agents are created. Once these agents are created the death threshold is too high to be met and the agents will never be destroyed resulting in higher population sizes that are not wanted, however as a byproduct the standard deviation approaches 0.

Population size is the second half of agent arrival rates. It was concluded that the different parameters all resulted in stable arrival rates within reasonable levels. However those arrival rates become stable due to increases in populations. In these simulations 168 agents were the target number of agents to maintain. The 65% thresholds with an expected arrival rate of fourteen agents per interval are the best settings in terms of trying to maintain initial population size as well as hitting the

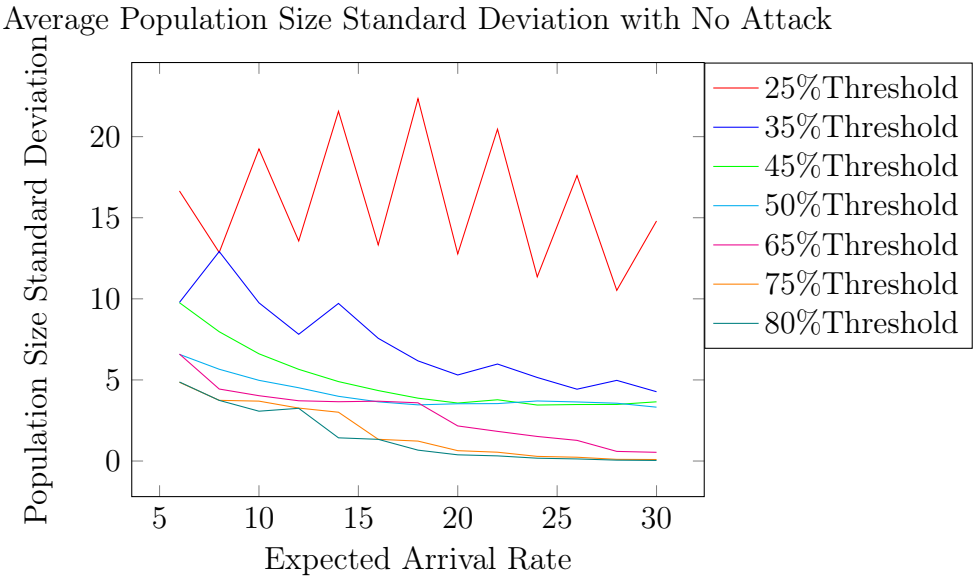

Figure 9.4: Population Standard Deviation as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

expected arrival rate.

9.3 Arrival Rates During Attacks Outside the Pheromone Map

It can be seen in Figure 9.5 that the average arrival rate becomes consistent after reaching an expected arrival of approximately 12 agents. Each average rate is raised slightly from the non-attack rate seen in Figure 9.1. During attack periods many agents get pulled into the pheromone map. As the agents are pulled in towards the target, hosts outside of the pheromone map become void of agents. At this moment more agents are created. This creation of agents continually happens each time the agents are pulled into the pheromone map. This can be seen by the raised standard deviation in Figure 9.6 versus the those in Figure 9.2. The higher standard deviation shows that the average arrival rate is changing between intervals but by a small

margin. Arrival rates change as populations change, and the population outside the map is continually changing as agents are pulled into the pheromone map.

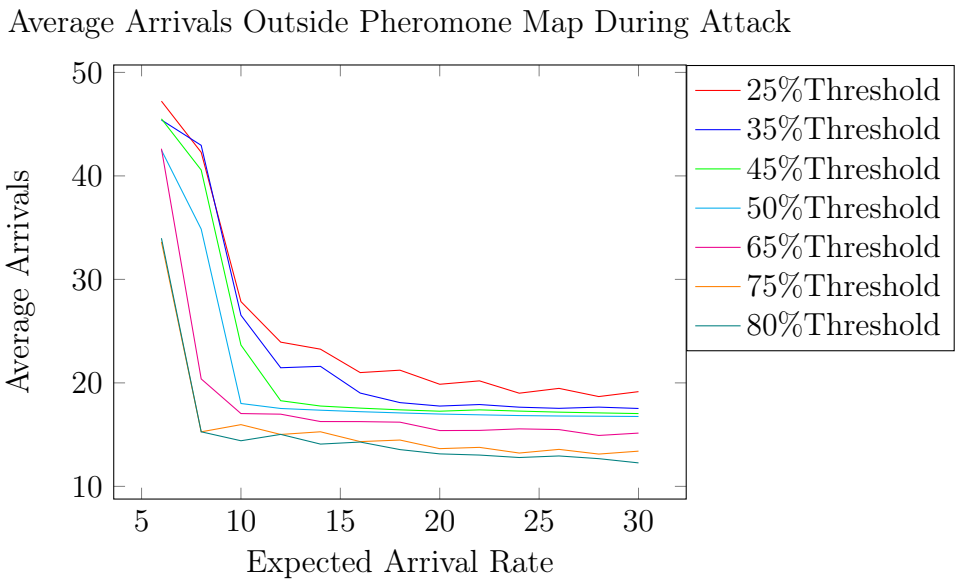

Figure 9.5: Average arrival rate of hosts outside the pheromone map during an attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

As was the case with no-attack the data point representing a 65% threshold while expecting 14 agents maintained an average arrival rate of 17, the lowest arrival rate and the closest to the expected arrival rate 14 while also maintaining a low standard deviation of approximately 3. The thresholds 0.75 and 0.80 had similar statistics but the thresholds appeared less stable during the non-attack phase.

9.4 Arrival Rates During Attacks in the Pheromone Map

The expected arrival rate can not be considered while inside the pheromone map because pheromone pulls agents towards the center of the map. More agents concen-

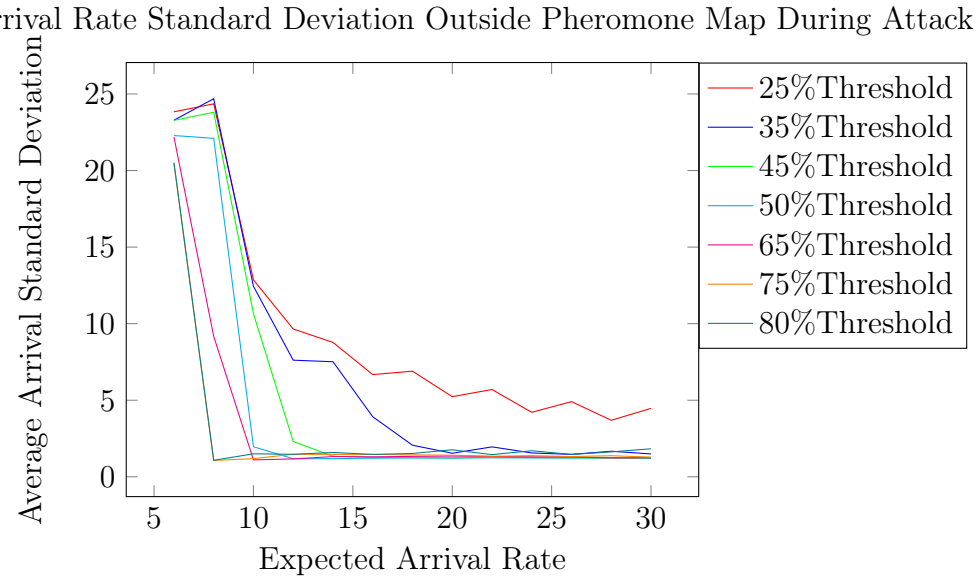

Figure 9.6: Average arrival standard deviation of hosts outside the pheromone map during an attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

trated on a set of hosts means those host will have higher arrival rates. Figure 9.7 shows that the average arrivals gradually decrease for all parameters. This behavior is expected because the birth and death of agents is turned off while a host is within the pheromone map and it was seen in non-attack Figure 9.1 that as the expected arrival rate grew that the actual arrival rate decreased. With no birth or death the population within the map should maintain fairly constant except for the addition of more agents that are generated outside the pheromone map which should be slowed by queue management. Agent arrival standard deviations have been increased to between 5 and 6. This is expected as agents are pulled into the pheromone map.

Unless the queue averages inside the pheromone map are extremely large than the average arrival rate is not that important. It is more important that queues do not reach the max queue size of 300. Figure 9.14 gives the average queue size within the pheromone map. From average queue size it can be stated that most queues in

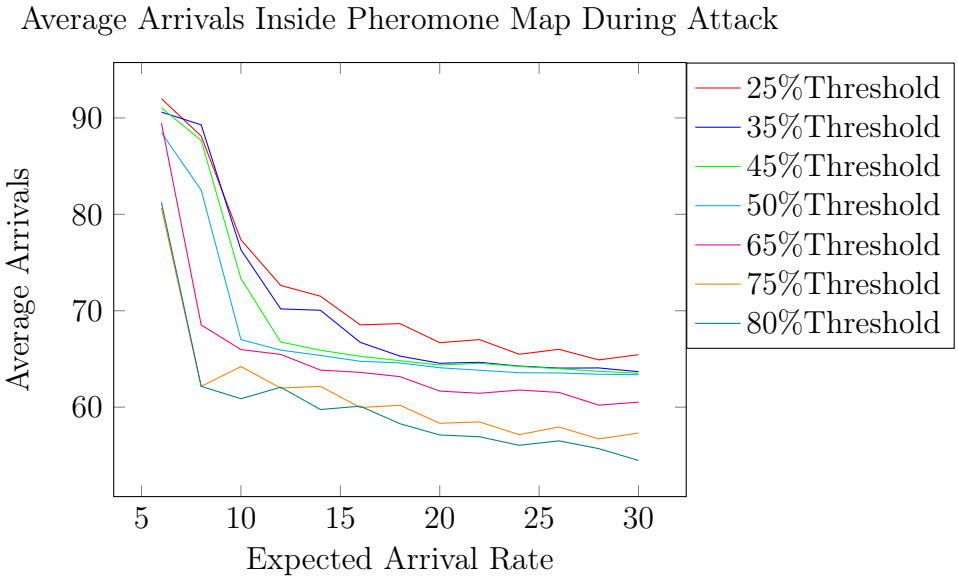

Figure 9.7: Average arrival rate of hosts inside the pheromone map during an attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

the pheromone map are larger than one, thus agents are waiting. However since the number of agents does not exceed the max queue size and visitation rates remain within the expected value, the change of parameters do not have much effect on the system within the pheromone maps. Parameters have much more of an effect on hosts outside of the pheromone map.

9.5 Population During Attacks

Figure 9.9 depicts the average population during attacks for different thresholds. It can be seen that any threshold with a low expected arrival rate creates populations over 3000 agents. These population sizes may have negative effects on an agent based system. Higher expected arrival rates maintain populations between 350 and 500. Although the thresholds 0.75 and 0.80 maintain the smallest populations during attacks other statistics have shown that those parameters are not optimal.

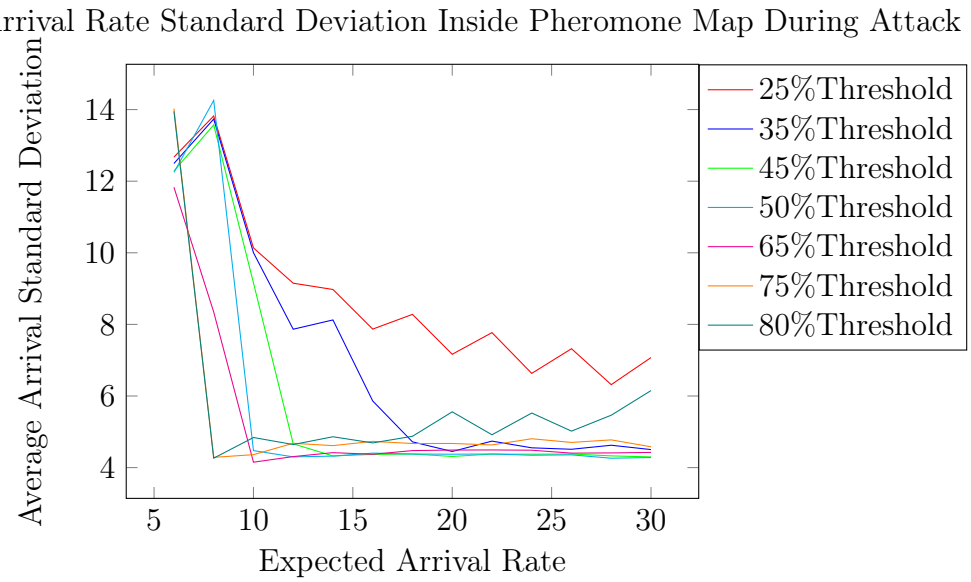

Figure 9.8: Average arrival standard deviation of hosts inside the pheromone map during an attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

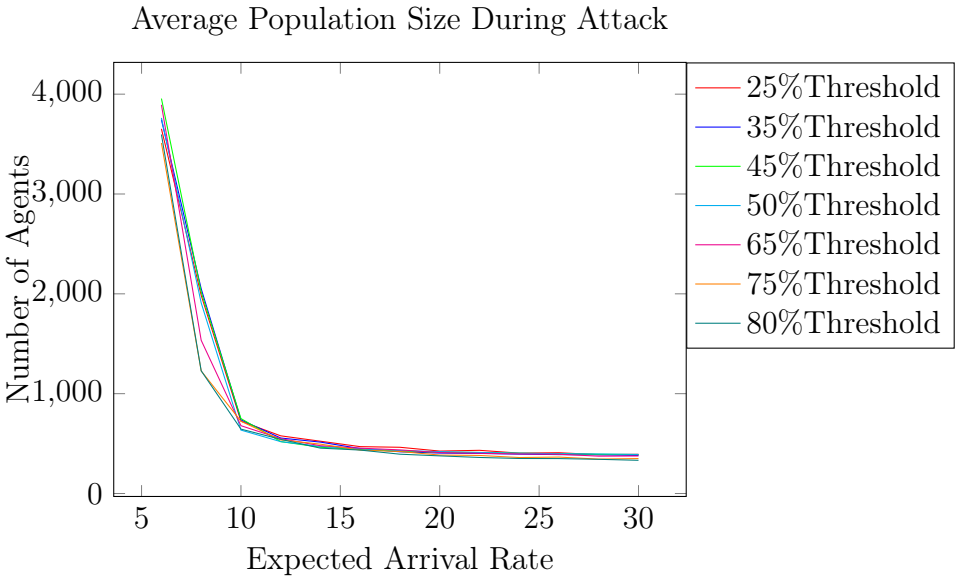

Figure 9.9: Population size during attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

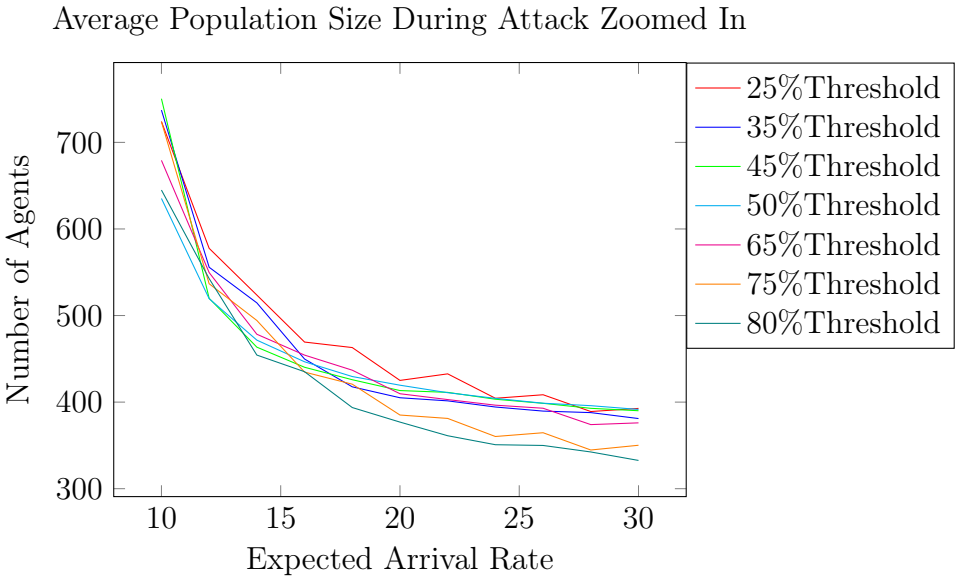

Figure 9.10: Zoomed in View of Population size during attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

The standard deviation of populations during attacks is shown in Figure 9.11. The standard deviations of the lowest two thresholds during high expectancy rates are between 50 and 200, indicating populations are often in flux and are not stable or reliable.

9.6 Queue Sizes

Queue size does not drastically change between any of the parameters during non-attack or attack both in and outside of the pheromone field. It can be noted that queues are shorter past the expected arrival rate of 10 for almost all thresholds. Two exceptions take place at the target host, both the 0.75 and 0.80 thresholds show increases as the expected arrival rate increases. However neither are troubling as the queue management algorithm never allows a queue to surpass an average of 20 agents.

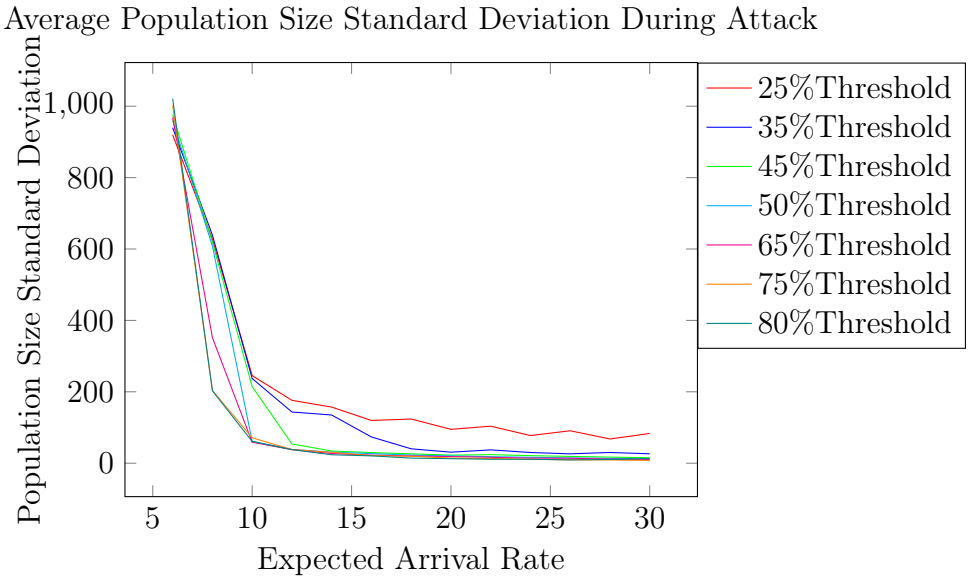

Figure 9.11: Population Standard Deviation during attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

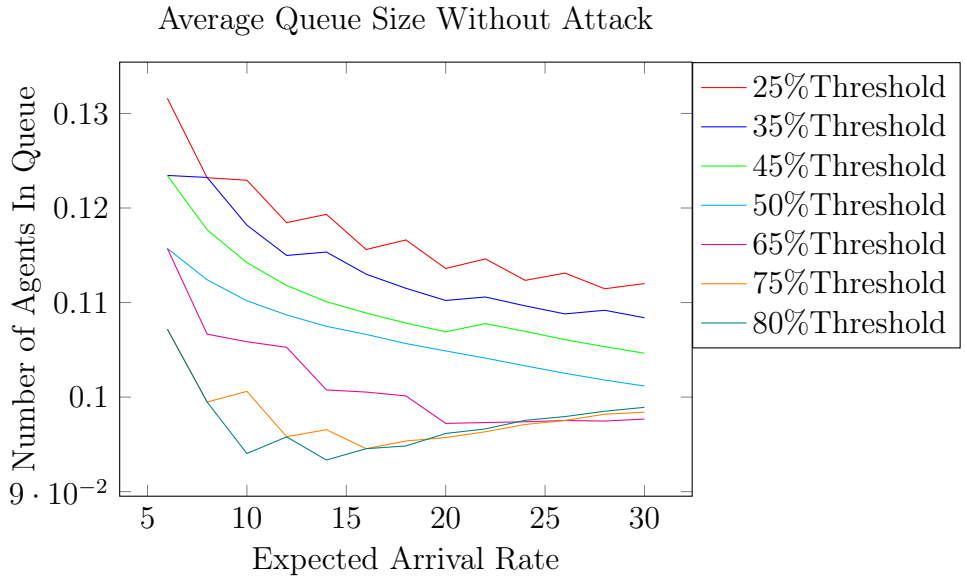

Figure 9.12: Average Queue Size without an Attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

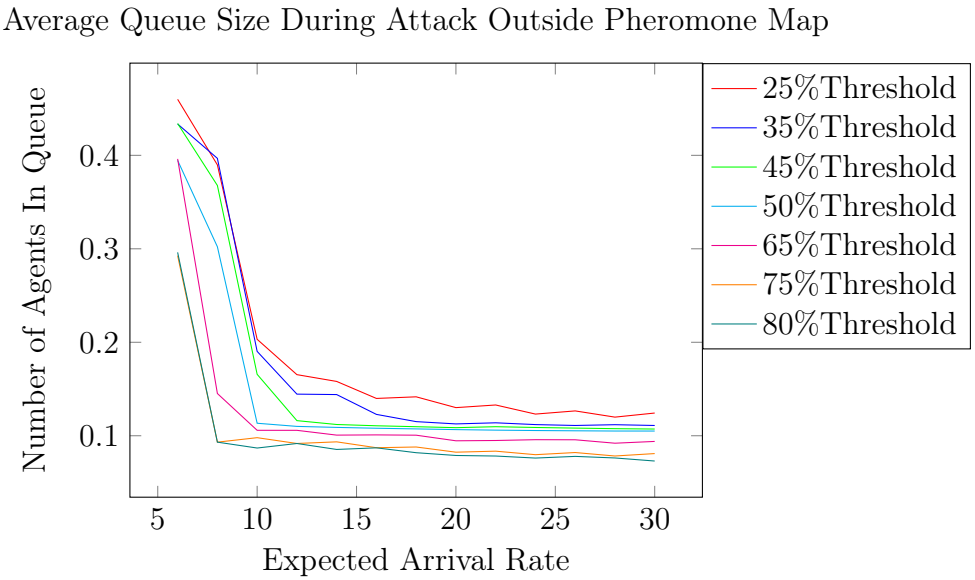

Figure 9.13: Average Queue Size without an Attack as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

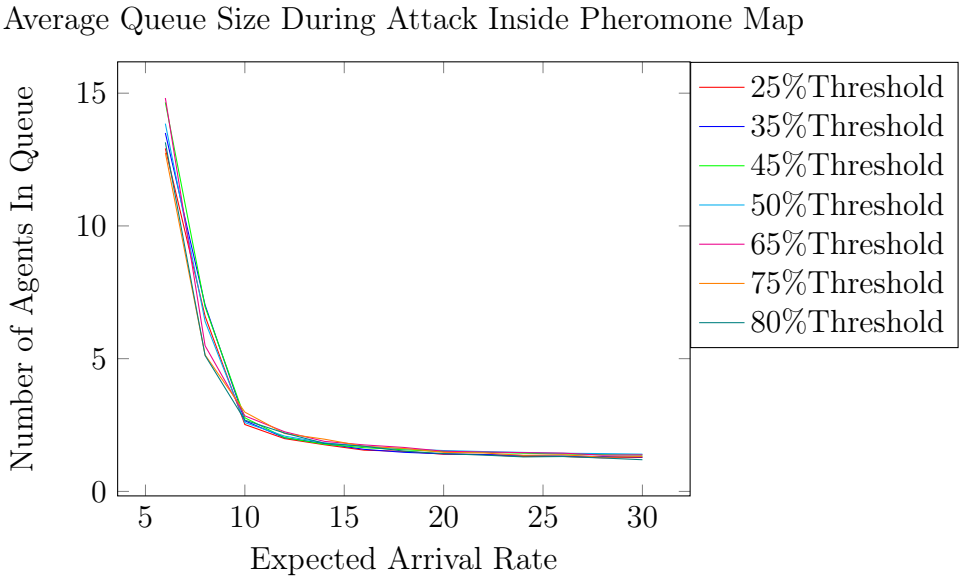

Figure 9.14: Average Queue Size during attack inside pheromone map as the expected Arrival Rate increases for a 41x41 toroidal grid at 7 different Birth Death Thresholds.

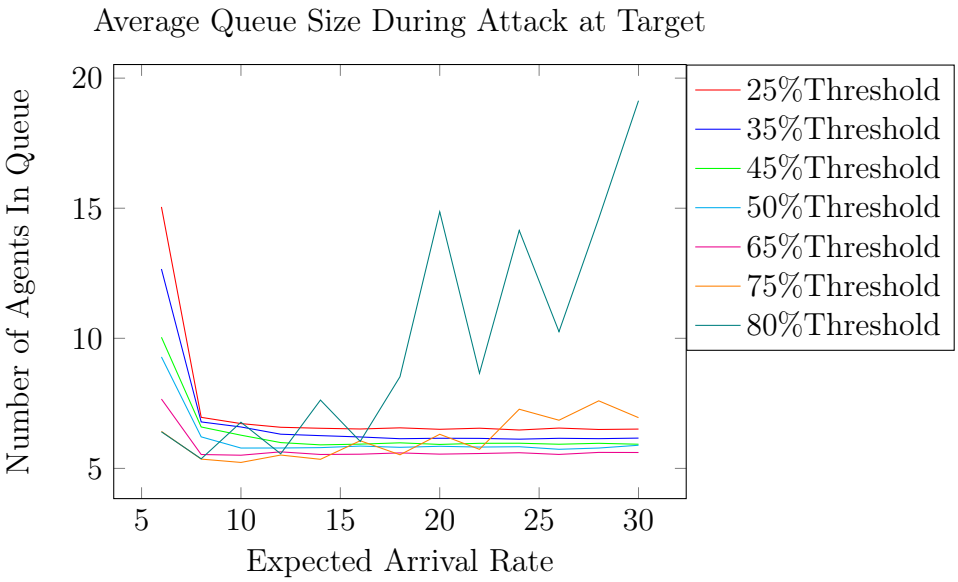

Figure 9.15: Average Queue Size during attack at target as the expected Arrival Rate increases for a 41x41 toroidal grid at death birth thresholds (percent difference between expected arrival and actual arrival) of 25%, 35%, 45%, 50%, 65%, 75%, 80%.

9.7 Conclusion of Parameters

After analyzing the varied statistics during attacks and periods of no attack it can be seen that it is the population during attacks within the non-pheromone map that parameters effect the most. The results show that low expected arrival rates cause short intervals that does not give the system time to adjust to change, causing over-corrections. The same can be said for both low death/birth thresholds along with high death/birth thresholds. Throughout the analyses of each statistic the parameter combination that performed well in each statistic was the 0.65 threshold with an arrival expectancy rate of 14.

Chapter 10: Scalability

To test scalability the settings with the best statistics from the previous experiments were applied to a 164x164 toroidal grid which has 16 times more hosts than the 41x41 toroidal grid. Table 10.1 shows that during the non-attack phase the arrival rate was 13.82, just under the expected arrival rate. The average population with no attack was on average about 8% higher than the starting average. Although with multiple hosts being able to create agents, this increase is not unexpected. Throughout all other experiments using population dynamics, the population over doubled during the attack phase. However in this scaling experiment the population stayed underneath double the starting population size. Average arrival rate within the pheromone map during attacks was 72.42. This average is not much higher than that of the 41x41 grid. Meanwhile the arrival rate outside of the pheromone map kept close to the expected arrival rate at 13.96. Average queue size also stayed relatively small through out all parts of the grid given the arrival rates throughout the pheromone map.

Figures 11.1 through 11.3 depict an example run on a 164x164 grid. The figures look very similar albeit on a larger scale than Figures 9.1 through 9.3. The most interesting figure is 11.2 which shows a reduction in the population towards 0 before recovering to normal values. It can not be confirmed what caused this drop in population. However the system was able to recover to expected levels and level out. This recovery is not a fluke as it can be seen that after the attack the population quickly declines to the same number of stable agents as before the attack.

Using parameters that work on a smaller grid successfully on a network 16 times larger is proof of the resiliency and scalability of the algorithms created to control the

Experiment Statistic	**Avg. (Std. Dev.)**
Arrival No Attack	13.82 (0.19)
Arrival During Attack Outside Pheromone	13.96 (0.22)
Arrival Average During Attack Inside Pheromone	72.42 (4.75)
Population Size After Attack	2742.84 (37.90)
Population Size During Attack	3216.16 (72.45)
Queue Size Without Attack	0.09 (0.00)
Queue Size During Attack Outside Pheromone	0.14 (0.00)
Queue Size During Attack Inside Pheromone	1.95 (0.43)
Queue Size of Target During Attack	6.57 (3.41)
Agent Age	159.25

Table 10.1: Statistics for 500 runs of the simulator with Death and Birth Implemented using a 164:164 toroidal grid, 10% agent population, expected arrival rate of 14 and a birth/death threshold of 50%.

arrival rates and population size of mobile autonomous networks.

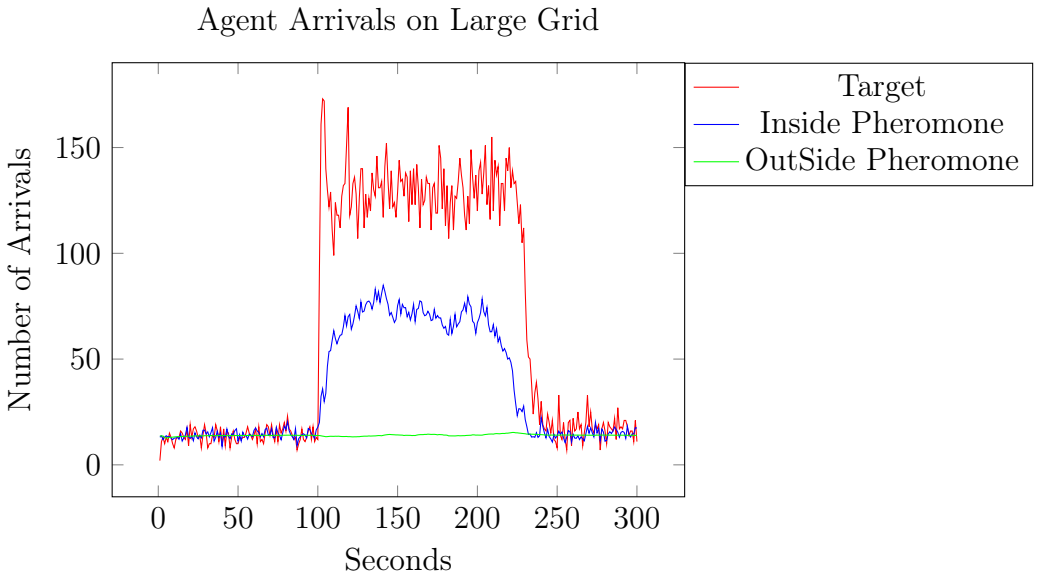

Figure 10.1: Average Arrivals for a Simulation using birth death and queue management using a 164x164 toroidal grid, 10% agent density, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack

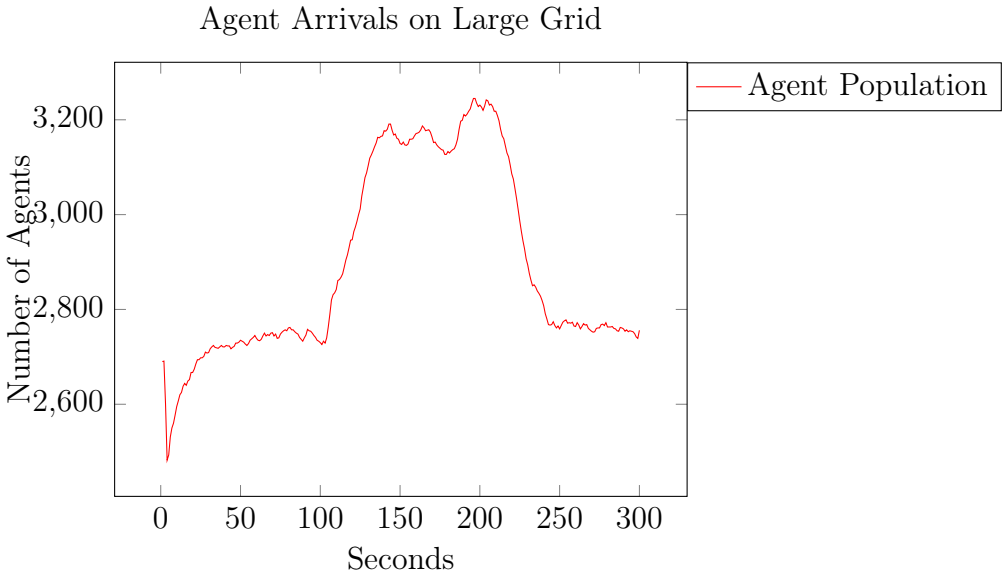

Figure 10.2: Average Population Size for a Simulation using birth death and queue management using a 164x164 toroidal grid, 10% agent density, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack

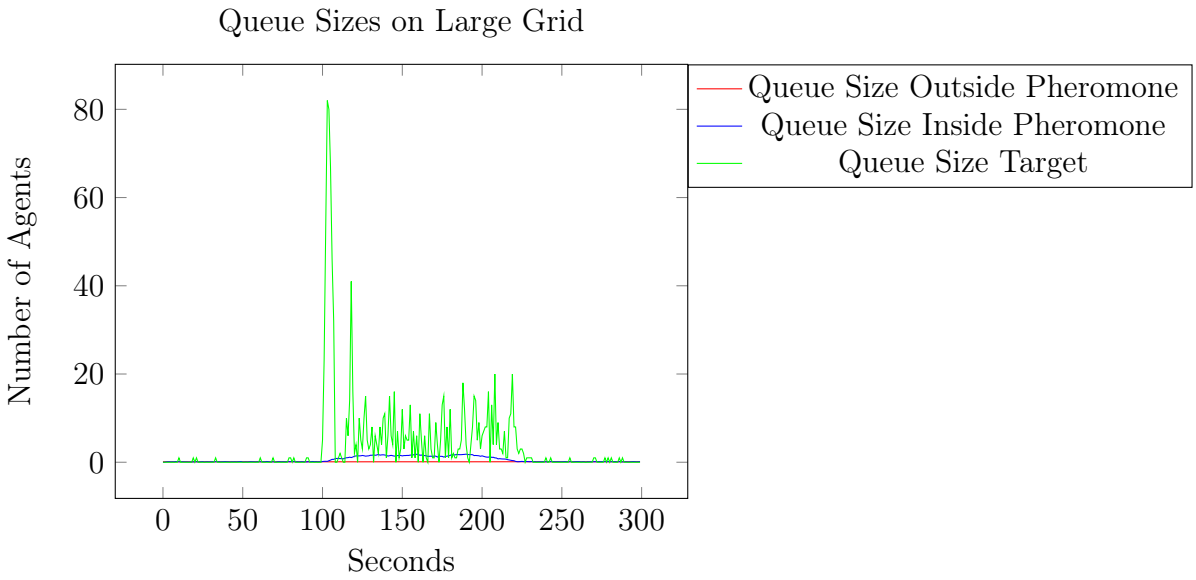

Figure 10.3: Average Queue Sizes for a Simulation using birth death and queue management using a 164x164 toroidal grid, 10% agent density, expected arrival rate of 14, and a birth/death threshold of 50%. The system runs normally for 100 seconds. Then an attack takes place between 100 and 200 seconds. From 200 to 300 seconds the system returns to normal after the attack

Chapter 11: Conclusions and Future Work

Using estimated arrival rates to manage both birth/death of agents along with managing queues allows for a successful and low overhead way to achieve the needed visitation rates to make autonomous mobile agent systems secure. It is clear that both death/birth of agents along with queue management is able to secure visitation rates throughout a mobile autonomous system and has the ability to adapt to different scenarios that may affect agent migration and movement throughout a network. It has also been found that use of such algorithms is scalable and perhaps adaptable to other systems outside of the Digital Ant Framework.

11.0.1 Differing Variables and Irregular Graphs

As variables in Equation (4.3) change for each individual host, the expected arrival rate at each node changes. When the difference in variables becomes large, the average arrival across all nodes has a standard deviation too large to be viable. It would be simple to adjust the population dynamic interval for each host, keeping a steady population across the system. However this strategy would lower the minimum number of arrivals at some hosts. Specifically, leaving hosts with fewer neighbors and higher transport times vulnerable to attacks. The easiest solution is to create virtual networks, but this requires a lot of overhead [13]. Irregular network topologies often have one highly connected host with multiple neighbors of small degree. In a virtual network this situation would require every agent to go through the transport queue of the highly connected host to reach each host's virtual neighbors. What effect would this delay have on the system?

11.0.2 Multiple Attackers

The algorithms in this thesis are capable of maintaining an average visitation rate across a network when a single host is attacked. Would the algorithm be able to maintain visitation rates with multiple attacks? Would multiple attacks during the same time span create too many agents such that the system would not be able to cope as seen with the implementation of the birth/death algorithm with no queue management? Would the birth rate need to be increased to keep up with multiple pheromone maps pulling in agents?

11.0.3 Multiple Types of Agents

As multiple types of agents are added to the system, it is reasonable to want to have the visitation rate of each type of agent meet a minimum visitation rate. However, to do so would require the management of each population separately and would require an increase of agents. As more agents are added to the system queues become longer. If agents of different types are not distributed evenly throughout queues, it is possible that the visitation rate of a single type of agent may drop and thus that agent type would be created, because that agent was stuck behind agents of different types. To manage such situations queue scheduling algorithms could be implemented, perhaps based on the rate of the types of agents processed.

11.0.4 Resiliency

Figure 10.2 shows the agent population drop close to 0 before rebounding to expected levels. In this thesis, agents were randomly distributed throughout the network, however if the agents all started on one host they would be killed off by queue management. However, would the system be able to create a new set of stable agents as the system did in Figure 10.2? In contrast if the agent started with full queues

at every host, would the system be able to reduce the number of agents to expected levels within a reasonable time frame?

Bibliography

[1] Fink, G.A.; Berenhaut, K.S.; Oehmen, C.S., "Directional Bias and Pheromone for Discovery and Coverage on Networks," Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference, pp.1-10 Sept. 2012

[2] McKinnon, A.D.; Thompson, S.R.; Doroshchuk, R.A.; Fink, G.A.; Fulp, E.W., "Bio-inspired cyber security for smart grid deployments," Innovative Smart Grid Technologies (ISGT), 2013 IEEE PES , pp.1-6 Feb. 2013

[3] Crouse, M.B.; White, J.L.; Fulp, E.W.; Berenhaut, K.S.; Fink, G.A.; Haack, J., "Using swarming agents for scalable security in large network environments," Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium, pp.1-4 Aug. 2011

[4] Kaizar, A.A.; Armin, M.R., "Dynamic agent population in agent-based distance vector routing," Second international workshop on Intelligent systems design and application, Dynamic Publishers, Inc., Atlanta, GA, USA, pp.95-200, 2002

[5] Konstantopoulos, C.; Mpitziopoulos, A.; Gavalas, Damianos; Pantziou, G., "Effective Determination of Mobile Agent Itineraries for Data Aggregation on Sensor Networks," Knowledge and Data Engineering, IEEE Transactions, vol.22, no.12, pp.1679-1693, Dec. 2010

[6] LaMonica, M., "Cisco: Smart grid will eclipse size of Internet" www.cnet.com/news/cisco-smart-grid-will-eclipse-size-of-internet, C-Net. Web. 18 May. 2009

[7] Kona, M.; Xu, C.Z., "A framework for network management using mobile agents" 16th International Parallel and Distributed Processing Symposium, Washington, DC, USA: IEEE Computer Society, pp.1-7, 2002.

[8] Marwaha, S.; Chen Khong Tham; Srinivasan, D., "Mobile agents based routing protocol for mobile ad hoc networks," Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE , vol.1, pp.1-7, Nov. 2002

[9] Deconinck, G.; Labeeuw, W.; Vandael, S.; Beitollahi, H.; De Craemer, K.; Rui Duan; Zhifeng Qui; Ramaswamy, P.C.; Vande Meerssche, B.; Vervenne, I.; Belmans, R., "Communication overlays and agents for dependable smart power grids," Critical Infrastructure (CRIS), 2010 5th International Conference, pp.1-7, Sept. 2010

[10] Jennings, N.R., "On agent-based software engineering," Artificial Intelligence, vol. 117, no. 2, pp.277-296, Mar. 2000

[11] Vidal, J., "Fundamentals of Multiagent Systems with NetLogo Examples," 24 Aug. 2007

[12] Prosser, B.; Dawes, N.; Fulp, E.W.; McKinnon, A.D.; Fink, G.A., "Using Set-Based Heading to Improve Mobile Agent Movement," Self-Adaptive and Self-Organizing Systems (SASO), 2014 IEEE Eighth International Conference, pp.120-128, Sept. 2014

[13] Dawes, N.; Prosser, B.; Fulp, E.W.; McKinnon, A.D.; "Using mobile agents and overlay networks to secure electrical networks," Eighth Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW '13), ACM, New York, NY, USA, pp.1-4, 2013

[14] Wu, Q.; Rao, N.; Barhen, J.; Iyenger, S.; Vaishnavi, V.; Qi, H.; Chakrabarty, K.,"On computing mobile agent routes for data fusion in distributed sensor networks,"Knowledge and Data Engineering, IEEE Transactions, vol. 16, no. 6, pp.740-753, June 2004

[15] Bakhouya, M.; Gaber, J., "Adaptive Approach for the Regulation of a Mobile Agent Population in a Distributed Network," Parallel and Distributed Computing, 2006. ISPDC '06. The Fifth International Symposium, pp.360-366, July 2006

[16] Beckmann, B.E.; McKinley, P.K., "Evolution of Adaptive Population Control in Multi-agent Systems," Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference, pp.181-190, Oct. 2008

[17] Fink, G.A.; McKinnon, A.D., "Effects of network delays on swarming in multi-agent security system," 1st International Workshop on Agents and CyberSecurity (ACySE '14). ACM, New York, NY, USA, pp.1-8, 2014

[18] Williams. B., "A Comparison of Static to Biologically Modeled Intrusion Detection Systems," M.S. Thesis, Wake Forest University, May 2010

[19] Fink, G.A.; Berenhaut, K.S.; Oehmen, C.S., "Directional Bias and Pheromone for Discovery and Coverage on Networks," Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference, pp.1-10, Sept. 2012

[20] Broder, A.; Kumar, R.; Maghoul, F.; Raghavan, P.; Rajagopalan, S.; Stata, R.; Tomkins, R.; Wiener, J., "Graph structure in the Web," Computer Networks, vol. 33, no. 1-6, pp.309-320, Jun. 2000

[21] Guido, C.; "Scale-Free Networks: Complex Webs in Nature and Technology," OUP Catalogue, Oxford University Press, Mar. 2007

[22] Berenbrink, P.; Cooper, C.; Elsasser, R.; Radzik, T.; Sauerwald, T.,"Speeding up random walks with neighborhood exploration," Twenty First Annual ACM-SIAM Symposium on Discrete Algorithms, pp.1422-1435. 2010

[23] Tanenbaum, Computer Networks (4th ed.), Prentice Hall Professional Technical Reference, 2002

Curriculum Vitae

Bryan Prosser

Education

Wake Forest University **2013-2015**

M.S. Computer Science

Wake Forest University **2009-2013**

B.S. Computer Science and Mathematics