Peer-to-Peer Network Modeling for Adversarial Proactive Cyber Defenses

by

Dennis Alberto Garcia

S.B., Computer Science M.I.T., 2016

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author .....
Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by .....
Una-May O'Reilly
Principal Research Scientist
Thesis Supervisor

Certified by .....
Erik Hemberg
Research Scientist
Thesis Supervisor

Accepted by .....
Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Peer-to-Peer Network Modeling for Adversarial Proactive Cyber Defenses

by

Dennis Alberto Garcia

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis implements a novel peer-to-peer network simulator that integrates coevolutionary algorithms in order to model adversarial attack and defense dynamics in networks. Modeling this behavior is desirable as it allows for network designers to better develop network defense strategies against adaptive cyber attackers. By developing a network simulator that implements a peer-to-peer protocol, we were able to control the environment and abstract away many of the complex details that would normally arise from using a live network. Because of this environment, we were able to design attack and defense models and grammars, construct arbitrary network topologies, and rapidly test adversarial behavior using the integrated coevolutionary algorithms. Second, the thesis implements the integration of the coevolutionary algorithms with a more complex, proprietary emulator that implements an advanced version of Chord. Our experiments with this system start to investigate the effectiveness of peer-to-peer networks as defenders as well as elucidate the issues of integrating coevolutionary algorithms in a real-world system.

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist

Thesis Supervisor: Erik Hemberg
Title: Research Scientist

Acknowledgments

The author would like to acknowledge his supervisors Una-May O'Reilly and Erik Hemberg. Both supervisors were amazing mentors and none of this work would have been possible without their constant guidance and help.

Contents

1 Introduction **13**

1.1 Motivation 13

1.2 Research Vision and Challenge 14

1.3 Contributions 15

2 Related Work **17**

2.1 Chord Protocol 17

2.2 Denial of Service Attacks 18

2.3 Adversarial Dynamics 18

2.4 Coevolutionary Algorithms 19

2.4.1 STEALTH 20

2.4.2 CANDLES 20

3 Methods **23**

3.1 Peer-to-Peer Networks: Chord 23

3.1.1 Chord Overview 24

3.2 Network Simulator 26

3.2.1 Logical Network 26

3.2.2 Logical to Physical Network 27

3.2.3 Chord Plus 28

3.3 Grammatical Representations of Attacks and Defenses 28

3.4 Coevolutionary Algorithms 30

4 Experiments **31**

4.1 Network Simulator Setup 31

4.1.1 Simulator Components 32

4.1.2 Logical to Physical Network Setup Differences 36

4.1.3 Chord Plus 37

4.2 Results 37

4.2.1 Logical Network Simulator Results 37

4.2.2 Logical to Physical Network Simulator Results 41

4.2.3 Chord Plus Results 42

5 Discussion **43**

5.1 Effectivness of Coevolutionary Algorithms 43

5.2 Effectiveness of Network Simulator 44

5.3 Thoughts on Network Emulation 44

6 Conclusions **47**

List of Figures

3-1 Physical network on the left and its virtual Chord overlay representation on the right. Also shown are the finger tables for nodes A, G, and F. 25

4-1 Topology 0: a very simple network used to benchmark the defensive actions for routing of shortest-path, flooding and Chord. 34

4-2 Topology 1: a larger network providing more nodes and a different topology. The increase in the number of nodes increases the search space significantly, thus eliminating the possibility of exhaustively searching the effects of all combinations of attacks. 34

4-3 Topology 2: a possible real network structure designed to simulate a more realistic mission. 35

List of Tables

4.1 Algorithm Settings 38

4.2 Network mission results for fixed attack, fixed defense and coevolution for topology 0 on the logical simulator implementation. 39

4.3 Network mission results for fixed attack, fixed defense and coevolution for topology 1 on the logical simulator implementation. 40

4.4 Network mission results for fixed attack, fixed defense and coevolution for topology 2 on the logical simulator implementation. 40

4.5 Network mission results for coevolution for all topologies on the logical to physical network simulator implementation 42

Chapter 1

Introduction

1.1 Motivation

Over the past couple of years, cyber attacks have increased in frequency, sophistication, and severity, and have been the cause of numerous disruptions in both industry and politics. Cyber attacks have become so common that it seems as if every week, there's news of a new major attack threatening the digital information of individuals, businesses, or even countries. The reason these attacks are even possible is because so much of our personal data and transactions now flow through networks, thus making it enticing for malicious entities to try and find ways to either intercept information as it flows through a network or disrupt the flow altogether. As a result, it is crucial for individuals and modern businesses to not only be aware of the capabilities of cyber attackers, but also to do everything in their power to build and maintain safe networks. The issue with the current state of cyber defenses, however, is that they are largely reactive in nature. For example, if some entity were to get attacked, that entity would most likely patch the vulnerability that led to it becoming exposed, after which the attacker would seek a different point of entry and begin this iterative process over again. This current inability of defenses to stay ahead of an adaptive attacker makes the task of creating secure, adaptive networks seem impossible. Thus, we must find a way to take steps to make this impossible task possible.

In order to achieve this, we have decided to test networks from an adversarial per-

spective in which a network evolves its defenses in order to mitigate the damage done by intelligent adaptive adversaries. The idea of modeling this adversarial behavior between two opposing entities has been realized in benchmark problems through the use of coevolutionary algorithms [7]. This example makes it clear that coevolutionary algorithms may be a good fit for modeling adversarial behavior in networks. Furthermore, among networks themselves, peer-to-peer networks are decentralized in nature and thus provide some natural defenses against cyber attacks. Thus, incorporating these coevolutionary algorithms into a peer-to-peer network also seems like a good fit.

While this idea may appear to be the best starting point, incorporating these coevolutionary algorithms into an established, live peer-to-peer network is incredibly challenging in itself as it can be very easy to get lost in the complexity of the system, making it difficult to integrate with coevolutionary algorithms. Finding a way around this complexity would allow us to abstract all extraneous details of a network and focus solely on the salient properties of a network and modeling adversarial behavior. This, in turn, would allow us to take a giant leap forward in our mission of creating secure, adaptive defenses. For this reason, the problem of both modeling a peer-to-peer network and integrating it with coevolutionary algorithms is the main motivation of this thesis.

1.2 Research Vision and Challenge

The vision of this thesis is the following: if we are to protect our information and our businesses in this digital age where cyber attacks are so common, then we must find a way to combat attackers who constantly change strategies by creating adaptive cyber defenses. In order to achieve this vision, the problem of deploying adaptive cyber defenses must be broken down into smaller problems. Because the task of integrating coevolutionary algorithms in a live peer-to-peer system is very complex, the first question we must answer is:

1. How can we effectively implement a peer-to-peer network simulator which in-

terfaces with coevolutionary adversarial genetic algorithms that need to manipulate defensive configurations of the network?

Then, after showing it is possible to model adversarial behavior in peer-to-peer networks using coevolutionary algorithms in a simulated environment, we can then focus on tackling our next question, which is:

2. How can we integrate the same coevolutionary genetic algorithms with a live peer-to-peer system with far more details and complexities and set out an example of evaluation?

The rest of this thesis focuses on explaining exactly how we answered these two research questions.

1.3 Contributions

The contributions of this thesis are the following:

- Designed and implemented a peer-to-peer network simulator that implements the Chord protocol [11] which allowed us to model simple network defensive behavior
- Integrated this network simulator with coevolutionary algorithms as part of a larger project named RIVALS [3] to support the simulation of adversarial behavior between cyber attacker and defender
- Created and ran experiments using RIVALS to show the effectiveness of peer-to-peer networks as a defender
- Incorporated one of the coevolutionary algorithms used in RIVALS into a novel proprietary software implementing an enhanced version of the Chord protocol and obtained initial results

Chapter 2

Related Work

In this chapter, we discuss a few of the works that we looked through and drew inspiration from throughout the project. It was through reading the work of these researchers that we were able to come up with many of the ideas presented in this thesis. In our search for literature that would be useful to us, we honed in on the following topics: peer-to-peer networks, distributed denial of service attacks, current uses of coevolutionary algorithms, and current technology that models adversarial behavior in some way.

2.1 Chord Protocol

One of the first things we looked at was the research done by [11] on the Chord protocol. This paper focuses on peer-to-peer systems and tackling the problem of decreasing the lookup time when attempting to locate specific data items. To address this problem, the authors of the paper created the Chord protocol, which is essentially a distributed hash table responsible for resolving keys to corresponding nodes that hold the data that needs to be retrieved. Throughout the paper, the authors provide a description of the protocol, arguments for its correctness and scalability, and pseudo code/examples of what a basic implementation of the protocol would look like. The reason we found this research so interesting and useful is because the pseudo code provided seemed simple enough to implement, which would potentially allow us to

set up a local version of a peer-to-peer protocol for rapid testing. However, the paper did not include every detail to go from the specifications to an implementation. I discuss the issues and challenges arising from this lack of clarity in section 3.2

2.2 Denial of Service Attacks

The second topic we sought to learn more about was on specific types of distributed denial of service (DDoS) attacks. DDoS attacks are a common way to disrupt certain network resources and are accomplished by flooding the target with a high volume of traffic. Because these attacks are so common, we decided to focus on defending against only these attacks rather than try and defend against every kind of cyber attack possible. In particular, we are interested in defending against the types of DDoS attacks described by [8] as they are more dangerous and malicious than normal DDoS attacks due to the difficulty level of detecting them. As described in the paper, these DDoS attacks are achieved by sending periodic DDoS streams that serve to exploit TCP's timeout mechanism. These DDoS streams come at regular frequencies and deter TCP flows heavily. Attacks like these are hard to detect because they can be sent in small waves and thus are not easy to spot amongst regular traffic patterns. Rather than detection of these attacks being our focus, however, our focus is resilience under a hard-to-detect, perhaps intermittent, DDoS attack. When designing the fitness functions discussed in the experiments chapter 4, we drew inspiration from this work by rewarding attacks on the network that can disrupt the network with minimal duration in order to mimic these hard-to-detect attacks.

2.3 Adversarial Dynamics

Although we seek to apply coevolutionary algorithms to a cyber security problem, it is worth discussing different approaches that have been taken within the field of Artificial Intelligence already to study adversarial dynamics in the context of cyber security. In [3], we provide a brief summary of these approaches that span the following

sub-fields of Artificial Intelligence: Game-Theory, Machine Learning, Evolutionary Computation, and AI-Planning. To start, Game-Theory has been used to find optimal game plan for conflicting adversaries. One such study that uses this approach is [12] where the authors study 'autonomous, collaborative control for resilient cyber defense' to distribute computational loads in dispersed clouds. In Machine Learning, [2] shows how trends learned from observational data can be useful in email spam-filters. Furthermore, the following three works emplore the use of Evolutionary Computation and focus on modeling adaptive systems in the context of cyber security: [4] analyzes botnet detection system and the effects of botnet evolution, [10] focuses on creating resilient infrastructures through competitive coevolution, and [6] uses coevolutionary strateiges to analyze a criticial infrastructure model. Finally, in AI-Planning, agents are able to devise strategies defined by goals. One work that falls under this sub-field of Artificial Intelligence is [1] where the authors study path re-planning for UAV's under critical conditions.

In addition to these works, another interesting example in which adversarial dynamics is used in the context of cyber security is the idea of Moving Target Defenses (MTDs). The idea behind MTDs is to coerce an attacker to change strategies constantly by making the defending system frequently change its state. The research conducted by [13] discusses evolving attackers using genetic algorithms that face adaptive defenders using MTDs.

2.4 Coevolutionary Algorithms

The next two works were examples of work that aimed to apply coevolutionary algorithms to a problem other than the typical benchmark problem in some way. These works were very influential in our decision to use coevolutionary algorithms to create robust network defenses.

2.4.1 STEALTH

STEALTH is the product of the work of [5] on the detection of tax non-compliance by using a coevolutionary genetic algorithm. By modeling the tax ecosystem as a network of nodes and edges and representing auditing processes as scores sheets, the researchers were able to simulate the coevolutionary behavior between tax evasion schemes and the auditing processes attempting to catch them. This research is nearly identical to the work we are trying to achieve as the auditors from the paper can be seen as the adaptive defending network we are trying to create, and the tax evasion schemes can be seen as the constantly changing cyber security attacks. The core of this research lies in the genetic algorithm used to coevolve both the tax evasion schemes and the auditors in order to find the best strategy for each entity. As a result, we believe that as long as we are able to create a similar framework where we can evaluate the effectiveness of cyber attacks on certain defenses of the network and vice versa, then we can simply run the same genetic algorithm to produce preliminary results.

2.4.2 CANDLES

The CANDLES system from [9], like STEALTH, greatly resembles the work we are trying to achieve in this thesis. CANDLES, Coevolutionary Agent-based Network Defense Lightweight Event System, is a framework that incorporates the use of co-evolutionary algorithms in order to competitively evolve strategies for attackers and defenders in the context of cyber security. Also similar to our work is the fact that the competitor action space of the framework is asymmetric. Although this work may seem to accomplish some of the research questions posed in this thesis, it differs in ways that still allows us a unique opportunity to fill in a specific niche in cyber security. The issue with CANDLES is that it is a very abstract network security simulation. Attacker solutions in this framework consist of a list of target machines, reconnaissance techniques, and exploits, while defender solutions use a paranoia threshold, a budget, a list of suspected targets, detection systems and dynamic mitigations. In

creating a novel network simulator, we distinguish ourselves by having a means of concretely measuring the results of simulating the arms race between attacker and defender using coevolutionary algorithms. Furthermore, we distinguish ourselves by making initial steps to incorporate these algorithms and simulate this behavior not just in the network simulator, but in a fully-developed peer-to-peer network system.

Chapter 3

Methods

In this chapter, we first expand on some of the design decisions behind choosing peer-to-peer networks as a starting point for creating robust and resilient defenses. Moreover, we also explain why we chose to work with the Chord protocol specifically, and provide a brief overview of how the protocol works. Then, we dive into the details of how the Chord protocol is implemented in the network simulator, how the simulator implementation differs from the actual protocol, and a few details of a proprietary emulator that implements an enhanced version of the actual Chord protocol. Finally, we end with a discussion on grammars and coevolutionary algorithms and how they fit in to the framework of this thesis.

3.1 Peer-to-Peer Networks: Chord

The main reason behind choosing a peer-to-peer network as a starting point in creating adaptive network defenses was that peer-to-peer networks have no single point of failure and are thus inherently more robust to defend against distributed denial of service attacks than other types of networks. As mentioned in section 2.2, distributed denial of service (DDoS) attacks are a class of cyber attack in which the attacker generates a large volume of traffic and directs it at a specific target. You can imagine a target being a specific server that is connected as part of a network with other servers. In flooding this server with a heavy volume of traffic, the server effectively

becomes useless and thus the network struggles to route traffic through this server. In a centralized network, this could be very damaging since a smart attacker could choose to attack the central server, or the server that is responsible for directing traffic to all the other servers, and wipe out the entire network that way. Peer-to-peer networks do not suffer from this issue. In peer-to-peer networks, data and resources are distributed and they are robust to topological changes such as nodes dropping out of service or nodes joining and leaving the network intermittently. This latter capability is provided in Chord via a stabilization routine. These are reasons as to why peer-to-peer networks provide an excellent starting point when considering setting up adaptive defensive networks.

While there were several peer-to-peer protocols that could have worked for this thesis, we chose the Chord protocol because it is efficient and extensible. Furthermore, the paper that presents the Chord protocol provides ample pseudocode. Although this pseudocode is at times vague and imprecise, it still allowed us to figure out how to take these specifications, and implement a model that runs on the network simulator.

3.1.1 Chord Overview

I now briefly describe a few of the important elements of the Chord protocol. Although this overview encapsulates many of the features the protocol has that we make use of in this thesis, I highly encourage the reader to read [11] if there is still any confusion left over the protocol. At its core, Chord is a key lookup service in the form of a distributed hash table. Chord aims to make the following process very efficient: given a key that holds some data or piece of information, find the node that has that key among a sea of peer nodes. Although this may seem simple enough to do, the mechanism that underlies this process is very intricate in order to meet the performance guarantees the authors' specify, which is essentially to have a logarithmic number of hops between any source node and destination node on the logical network representation. To start, Chord assigns each node joining the network and each key m-bit identifiers, and places each identifier in the identifier circle. We illustrate this hashing and placement process in Figure 3.1. In this figure, each node in the physical network on the left is given

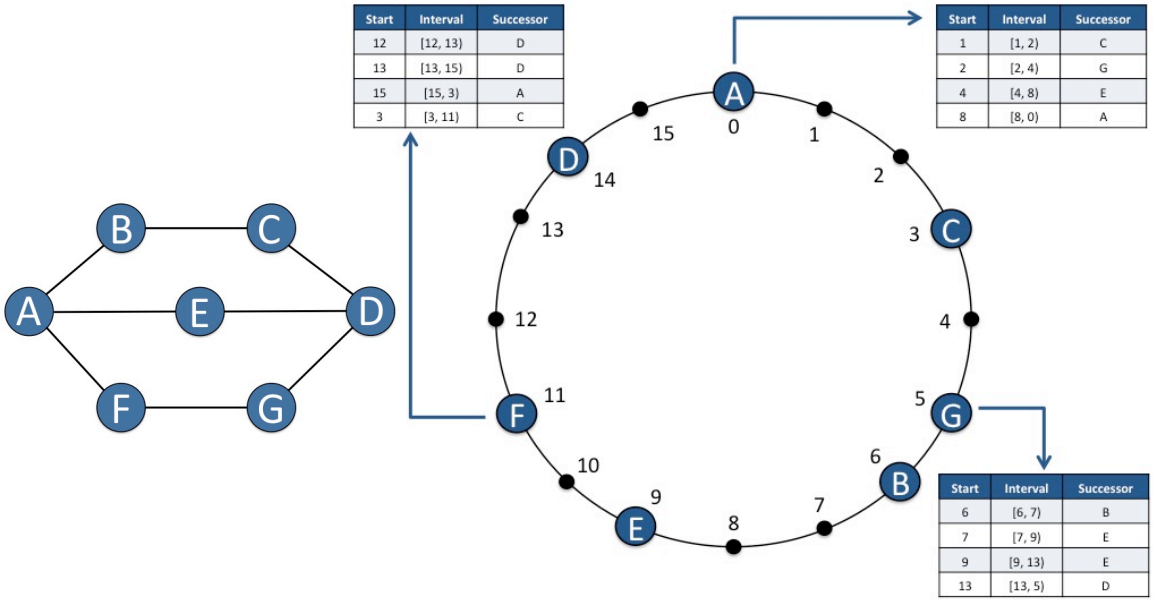

Figure 3-1: Physical network on the left and its virtual Chord overlay representation on the right. Also shown are the finger tables for nodes A, G, and F.

an identifier and logically placed, thus becoming a virtual node in a location on the identifier circle. Then, each key in the identifier circle is assigned to a node. The node that the key is assigned to is the node that most immediately follows the spot where the key is hashed to in the circle. For example, in Figure 3.1, if a key were given the identifier of 2, the node responsible for this key would be the node at identifier 3 since that is the closest node clockwise of the key's identifier. Furthermore, if a key were given the identifier of 3 instead, the node responsible for this key would still be the node at identifier 3, as it is already on the same identifier as a node. Each node maintains extra information in an individual lookup table called a finger table that it uses to direct key queries efficiently along the circle. A lookup sends the query at least halfway to its destination by taking advantage of the information stored in the finger table. Furthermore, Chord handles nodes entering and dropping off the network by updating node finger tables periodically as well as reapportioning keys around the network as needed. This stabilization naturally lends itself to adversarial situations where an adversary intentionally forces nodes out of availability. While the main features of Chord are advantageous and reasons why we chose to work with this protocol, this ability to handle nodes joining and leaving the network gracefully is

what attracted us most to it.

3.2 Network Simulator

The centerpiece behind the network simulator we implemented is an implementation of the Chord protocol. The network simulator has two versions. One version, we describe as the 'logical version', and the other, we describe as the 'logical to physical' version. The differences between these two versions of the software are explained in the next two subsections. Before moving there, however, we first highlight the differences between our implementation of Chord, and the definitive specification of the protocol that [11] describes. While these are the differences that stand out, there may be in fact more subtle differences since at times, the specifications given in the paper were unclear.

To start, my implementation currently simply models Chord on a single workstation. In addition to this, upon nodes leaving or joining the network, the original Chord protocol eventually stabilizes itself through periodic actions. In contrast, in the network simulator implementation, every time a node leaves or joins the network, successor and predecessor pointers as well as the finger tables are immediately repaired. Another contrast is that nodes in the Chord network become part of the circle by receiving an m-bit identifier obtained by hashing the nodes with SHA-1. In our implementation, we use Python's built-in random library to provide the identifiers instead.

3.2.1 Logical Network

Continuing from the discussion earlier on the two versions of the network simulator we implemented, we now describe the 'logical' version of the simulator. To provide a bit of context before we dive into the details, let's refer back to Figure 3.1. Like we mentioned before, in this figure, we have a simple physical network on the left, and the virtual Chord overlay network that gets constructed by the protocol on the right. In this example, recall that if we were interested in finding a specific key in the network,

we can ask any peer, and that peer would use its finger table information to route our query to some peer that is closer to the target peer containing the desired key in the identifier circle. This series of queries provides a hopping pattern of the nodes pinged along the way before reaching the target node with the key. For example, if we ask node F where the key with identifier 3 is, it would pass our query to node A, and then node A would find that they key is located at node C. This results in a hopping pattern of F, A, C. In the simulator implementation, rather than use the protocol as a means of finding a key in the network, we use it as a means to represent sending a message through the network. We achieve this by asking the peer we consider the starting node, or the node responsible for sending the message, to find the identifier associated with the target node. In this sense, the difference is that we now use the protocol to lookup target node identifiers instead of key identifiers. The reason we name this version the 'logical' version is because we assume the message gets sent via the hopping pattern this lookup incurs through the virtual representation of the physical network rather than through the actual physical network itself.

3.2.2 Logical to Physical Network

The key difference between the logical version and the logical to physical version of the simulator is that the logical to physical version increases the complexity and reality of the simulator by simulating messages flowing through the physical layer of the network as opposed to just through the virtual Chord overlay representation. As a result, in this version, when sending a message, instead of modeling this as the message hopping through the Chord network and reaching its destination, each hop from one node to another in the virtual representation represents a message passing from the equivalent nodes in the actual physical representation of the network. The reason this increases the realism of the network simulator is because this is how the Chord overlay would actually be used in the real world.

3.2.3 Chord Plus

With these two versions of the network simulator, we were able to come close to simulating a real network and perform rapid testing without actually having to use a real distributed system. Towards the end of the project, however, we were able to obtain proprietary software that implements an emulation of the fully distributed version of the Chord protocol with significant enhancements. This software is incredibly complex and uses the most current technology to implement the protocol. Because this software is proprietary and still under development, the details of the specifications of the system or of the implementation are unavailable. However, we can report that we were able to successfully interface with the software through the use of configuration files and by parsing output files to get metrics regarding the network capacity. This is the final version of the Chord protocol we ended up with and had the opportunity to run very basic initial experiments on.

3.3 Grammatical Representations of Attacks and Defenses

To support the coevolutionary algorithms that we integrated with the network simulator to be able to generate populations of random attacks on the network and tell a given network which defense to use, we had to make use of grammatical representations. An example of the grammar for generating simulated distributed denial of service attacks in the logical version of the simulator upon receiving the start symbol, `<Attacks>`, is:

⟨*Attacks*⟩ ::= DDoSAttack(⟨*node*⟩, ⟨*start_time*⟩, ⟨*duration*⟩)
| DDoSAttack(⟨*node*⟩, ⟨*start_time*⟩, ⟨*duration*⟩), ⟨*Attacks*⟩

⟨*node*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6

⟨*start_time*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨*duration*⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

One of the important features of this grammar to note is that a DDoS attack is represented by specifying a node to attack, a specific time to begin the attack, and the duration of the attack. Also, the grammar is recursive, thus allowing for the possibility of generating attacks on several nodes as opposed to an attack on just one. An example of the attack grammar used in the logical to physical version of the simulator upon receiving the start symbol `<Attacks>` is:

⟨*Attacks*⟩ ::= 'physical_attacks': [⟨*physical_attacks*⟩], 'logical_attacks': [⟨*logical_attacks*⟩]}

⟨*physical_attacks*⟩ ::= DDoSAttack(⟨*node*⟩, ⟨*start_time*⟩, ⟨*end_time*⟩), ⟨*physical_attacks*⟩
| DDoSAttack(⟨*node*⟩, ⟨*start_time*⟩, ⟨*end_time*⟩)

⟨*logical_attacks*⟩ ::= DDoSAttack(⟨*node*⟩, ⟨*start_time*⟩, ⟨*end_time*⟩), ⟨*logical_attacks*⟩
| DDoSAttack(⟨*node*⟩, ⟨*start_time*⟩, ⟨*end_time*⟩)

⟨*node*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6

⟨*start_time*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨*end_time*⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

The difference here is that because both the physical and virtual networks are utilized to represent the flow of a message, an attacker is allowed to specify nodes at the physical and virtual layers to attack. The grammar remains recursive, as before, in nature, and stores the generated list of attacks in a dictionary. The defense grammar in both versions of the simulator is simple as the grammar just chooses between three routing mechanisms, or defense protocols. An example of this grammar given the start symbol `<Defense>` is:

⟨*Defense*⟩ ::= shortest_path_protocol | flooding_protocol | chord_protocol

More details about the routing mechanisms a defender can choose from and how attacks in general are represented are discussed in the next chapter when we discuss our experiments.

3.4 Coevolutionary Algorithms

Coevolutionary algorithms feature two populations competing with each other and adapting through the process of selection and mutation. These algorithms exhibit very complex, adaptive behavior. For more details, see the discussion on the five different types of these algorithms used in [3]. These are the five algorithms we integrate with the network simulator. The important aspect of coevolutionary algorithm integration that this thesis dealt with was the design task of logically connecting the behavior of the peer-to-peer network to the action space defined by a grammar used by all of the coevolutionary algorithms. For example, the defensive grammar mentioned in the previous section gives attackers the choice among "all paths", "shortest path with retries" and "Chord". As a result, the challenge with the algorithms was to support each of those mechanisms and allow the algorithms to direct the network to behave in one of those ways. In addition to this, another challenge was collecting network simulation outcomes and "passing" them to the coevolutionary algorithm in order to calculate overall fitness.

Chapter 4

Experiments

In this chapter, we discuss and analyze the experiments that we conducted once we fully integrated the network simulator with the coevolutionary algorithms described in the previous chapter. The experimental results, descriptions, discussion, and analysis that follow were originally stated in [3] so credit for them is shared with the co-authors of this paper [1]. Overall, these experiments help us seek to understand how well we can model adversarial behavior in networks. We first detail some of the steps we took in setting up these experiments, and then discuss the results of running the experiments across both versions of the simulator and on the proprietary Chord emulator.

4.1 Network Simulator Setup

While chapter 3 went into the details of the network simulator, there are some experiment-specific components that we implemented into the simulator on top of the Chord protocol in order to be able to run experiments successfully. In this section, we give an overview of the context, terms and additional components of the network simulations including: the network topology, missions, attacker goals and capabilities, and defender goals and capabilities. Although most of these components are the same across the different versions of the simulator, there are subtle differences

[1] Some text is verbatim from the paper, and other text has been updated to reflect refinements to the experimentation and implementation

and these are highlighted and discussed.

4.1.1 Simulator Components

Context: Briefly mentioned during the discussion of grammars in chapter 3, a distributed denial of service attack is a common technique used by cyber attackers to disrupt a specific network's resources by sending large amounts of traffic to that network. The simulator models these attacks by using the attack grammar specified in section 3.4 and the coevolutionary algorithms to generate attacks on random nodes.

Missions: To best model the arms race between a cyber attacker and cyber defender, we included the concept of a mission in the network simulator so that the attackers and defenders each had a goal to achieve. A mission is comprised of a series of tasks that the defender tries to complete. A task consists of a start node, an end node, and a maximum time limit during which the defender can complete the task. For now, tasks are modeled as sending a message through the network from the start node to the end node. In reality, tasks could be more complex operations such as using Internet Relay Chat (IRC), or transferring a file using the File Transfer Protocol (FTP) from the start node to the target node. If the defender does not complete a task on time, the task fails. If any task fails in a series of tasks, the mission fails. For simplicity in these experiments, we have limited the number of tasks in a mission to be one.

Attacker: The way in which an attacker is modeled in the simulator has already been discussed both in the context paragraph and in section 3.4. The goal of an attacker is to disrupt the flow of traffic in the network with as little effort as possible. An attacker may achieve this by choosing nodes in the network to launch DDoS attacks against in order to cause mission failure. As seen from the grammatical representation of an attacker, attackers are powerful in the sense that they can specify any node in the network to attack, the time in which the attack starts, and the duration of the attack.

Defender: The goal of a defender is to ensure mission success in the face of constant cyber attacks. A defender attempts to achieve mission success by choosing among three different routing configurations.

- **Shortest path protocol**: The network calculates the shortest path between the start node and target node and sends a packet along this path. If at any point along the network, the path gets blocked via a node failure caused by an attacker's DDoS attack, then the network waits until the path is free to send the packet on its way again.
- **Flooding protocol**: The flooding protocol works by sending a copy of the packet along all possible paths from the start node. If a path gets blocked due to node failure from an attack, it stalls like in the shortest path protocol. The difference, however, is that other paths may be available for the packet to travel through. Once the first instance of the packet shows up at the destination, the task is completed.
- **Chord protocol**: The implementation of the network simulator of the Chord protocol serves as the network's last routing mechanism. In the logical version of the simulator, we model the node reaching its destination via the hops specified by the node finger tables through the virtual Chord representation. Attacks on a node of the virtual representation are modeled by that node leaving the Chord ring, and coming back when the attack is finished.

Network Topology: Another capability of the simulator is the ability to create network topologies. During most of our initial testing, we used the simple topology from Figure 4.1 in order to exhaustively explore simple mission scenarios and determine the correctness of both the simulator and the coevolutionary algorithms. In order to make the problem more complex and realistic, we scaled up the topologies to the ones in Figures 4.2 and 4.3 so that the search space would be much too large to be able to exhaustively search over all possible attack scenarios.

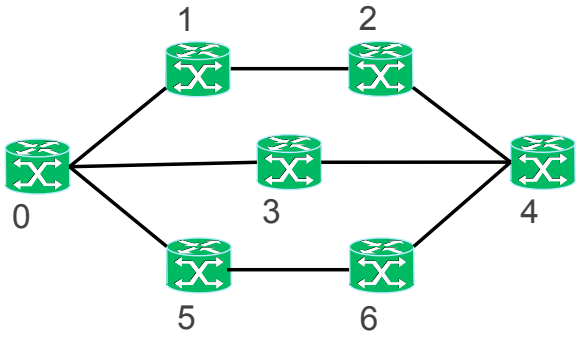

Figure 4-1: Topology 0: a very simple network used to benchmark the defensive actions for routing of shortest-path, flooding and Chord.

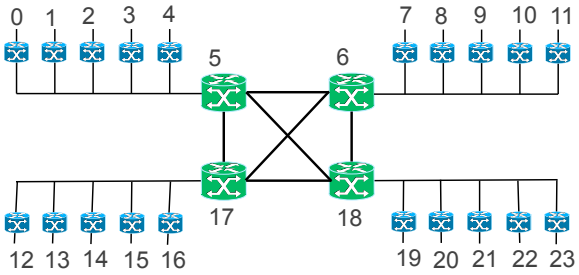

Figure 4-2: Topology 1: a larger network providing more nodes and a different topology. The increase in the number of nodes increases the search space significantly, thus eliminating the possibility of exhaustively searching the effects of all combinations of attacks.

Fitness Functions: Fitness functions are the method we use to evaluate how well an attacker or defender performed. In the case of an attacker, we reward attackers who are able to disrupt a mission by attacking as few nodes as possible for short amounts of time. Furthermore, we penalize attackers who succeed in disrupting a mission, but have to achieve this goal through many attacks for long durations. This reward system for an attacker is captured by the following fitness function:

$$f_a = \frac{1 - mission\_success}{(n\_attacks \cdot total\_duration) + n\_attacks}$$

In this function, $mission\_success$ describes whether the mission succeeded (1) or failed (0), $n\_attacks$ is the total number of nodes attacked in the network, and $total\_duration$ is the aggregated amount of time nodes were under attack. The extra $n\_attacks$ term in the denominator molds the fitness function to

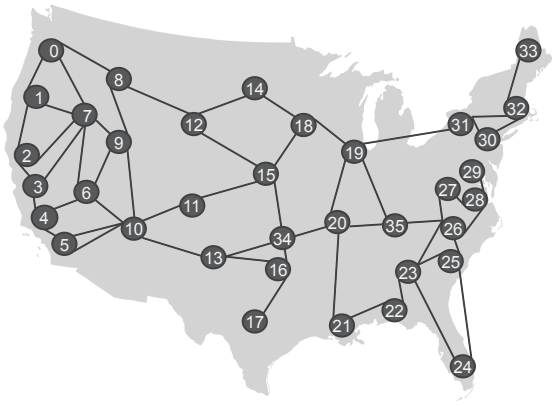

Figure 4-3: Topology 2: a possible real network structure designed to simulate a more realistic mission.

reward attacks consisting of the least number of attacks.

In the case of defenders, we reward defenders who complete the mission, and also reward those who are able to do this with a low number of hops. The motivation behind this is to punish defenders for using up network resources. For example, if the network decides on using the flooding protocol, this defense will most likely succeed in transmitting the packet, but will incur a large penalty for using too many network resources. This reward system for defenders is captured by the following fitness function:

$$f_d = \frac{mission\_success}{overall\_time \cdot n\_hops}$$

Here, $overall\_time$ is the total amount of time a specific routing protocol took to complete the mission, and n_hops is the total number of hops taken by the protocol to complete the mission. Finally, the last assumption made in the network simulator is that any edge cross between nodes in the topologies or in the virtual chord representation are of unit length and thus take one timestep to traverse.

4.1.2 Logical to Physical Network Setup Differences

When compared to the setup of the experiments for the logical simulator, the physical to logical simulator had only two key differences.

Chord protocol: As discussed in chapter 3, the implementations of how messages are passed through the network differs between the two versions of the network simulator. In this version, each hop in the virtual representation corresponds to hops in the physical network. If a node finds it cannot make a hop to another node in the ring because of attacks at the logical or physical layer, it simply scans through its finger table to see what node is available that produces the largest hop. Furthermore, attacks on a node in the physical network renders it useless in the physical network and gets represented in the virtual network by means of leaving the network.

Fitness functions: Because both logical nodes and physical nodes are available for attack in this version of the simulator, the way in which we reward attackers changes. The new fitness function for attackers is:

$$f_a = \frac{1-mission\_success}{(2 \cdot n\_physical \cdot p\_duration)+n\_physical+(n\_logical \cdot l\_duration)+n\_logical}$$

In this equation, $mission\_success$ still represents whether the mission succeeded (1) or failed (0), $n\_physical$ describes the number of attacks launched on nodes in the physical layer, $n\_logical$ represents the number of attacks launched on nodes in the virtual layer, $p\_duration$ represents the total aggregated time nodes in the physical layer were under attack, and $p\_logical$ represents the total aggregated time nodes in the logical layer were under attack. When taken as a whole, this fitness function penalizes attacks launched on the physical layer more heavily because taking out a node in the physical layer has deeper consequences than taking out the corresponding node in the virtual layer.

4.1.3 Chord Plus

While these components, definitions, and formulas may have worked when using our own simulator, we had to throw much of this out the window when seeking to work with the proprietary Chord emulator as the emulator had a completely different way to model attacks, routing configurations, etc. It was mostly through inspecting the code, and working with configuration and output files that we were able to run a very basic integration test with the coevolutionary algorithms.

4.2 Results

We ran all five of the coevolutionary algorithms discussed on the three different topologies for the logical implementation of the simulator as well as for the physical to logical implementation. The parameters used for the algorithms are taken from Table 4.1 which was taken from [3]. The results shown are those averaged over 30 runs of the algorithms. For the proprietary emulator, we managed a very basic, though successful, integration of the coevolutionary algorithms with the live network system and report interesting findings. The main idea behind these experiments is that in using the grammars to generate random populations of attacks, the coevolutionary algorithms should evolve both the attacker and defender. Then, the algorithms should be able to state the individual with the best fitness at the end of the runs. For the attacker, this means the algorithms should find which attack was most effective on the network given the fitness function we specified for attackers. Similarly, for the defender, this means the algorithms should find which network routing configuration proved most effective based on the defender fitness function at withstanding attacks.

4.2.1 Logical Network Simulator Results

After incorporating the extra components specified in section 4.1.1 into the logical network simulator implementation, we incorporated the coevolutionary algorithms, and ran all five algorithms discussed. Before we made these runs, it was our hypothesis

Table 4.1: Algorithm Settings

Parameter Setting	Compare-on-one	Network Simulations
Population size	10	40 (10 for Topology 2)
Archive size	10	20
Generations	1000	20
Max length	10	20
Parent archive probability	0.9	0.9
Crossover probability	0.8	0.8
Mutation probability	0.1	0.1
Mutation bias low	-0.15	NA
Mutation bias high	0.1	NA
Generation loop breakout	500	NA
Grammar	No	Yes

that because the Chord protocol has a way of stabilizing the network upon nodes leaving the network and finding ways to hop to the destination regardless, that it had the best chance of the three routing configurations at withstanding the attackers. Also, it is worth noting that before we ran the algorithms across all of the topologies we generated, we first performed an exhaustive search over topology 0 to argue about the correctness of both the implementations of the routing protocols and the algorithms.

In this exhaustive search, we generated all possible combinations of nodes to attack where each attack lasted the full duration of a specified task. We found that when using the **shortest path protocol**, the defender failed to complete the mission when there was an attack along any of the nodes in the shortest path. For example, if a task of the mission in topology 0 was to send a message from node 0 to node 4, then we saw an attacker disrupt the mission by attacking any combination of the nodes 0, 3, or 4. Furthermore, we observed that for the **flooding protocol**, the set of attacks that blocked a message going through were those that blocked nodes along all paths from the start node to the target node. For example, we saw that to block a defender from sending a message from node 0 to node 4, an attacker had to block nodes 1, 3, and 5 at the same time or similar combinations that block all paths. Both of these observations were in alignment with our expectations. Finally, we observed that out of all of the possible attacker combinations, the only one successful against blocking Chord was an attack which knocked out all the nodes in the network. This was as expected as well because of Chord's ability to handle node failure and pass messages along the network regardless of nodes leaving the network. Chord succeeds in this

Table 4.2: Network mission results for fixed attack, fixed defense and coevolution for topology 0 on the logical simulator implementation.

Algorithm	**Attack**	**Defense**	**Coevolution**
Coev	1.000 ± 0.000	0.250 ± 0.000	0.227 ± 0.05
MinMax	0.975 ± 0.000	0.250 ± 0.000	0.200 ± 0.060
MaxSolve	0.826 ± 0.014	0.207 ± 0.088	0.263 ± 0.159
IPCA	0.690 ± 0.079	0.250 ± 0.000	0.333 ± 0.063
rIPCA	0.698 ± 0.082	0.250 ± 0.000	0.463 ± 0.018

because in the case that the target node is attacked, the protocol designates a new target node by means of passing along its keys and content to its successor node, thus keeping the target node alive.

Because this exhaustive search is impossible on the larger topologies, we ran the algorithms across all topologies with the parameters and number of runs specified earlier. The results of these runs can be found in tables 4.2, 4.3, and 4.4. To explain the tables, the values presented are the average fitness values of the last generation over the 30 runs. Across all topologies the *Attack* column means we fixed the defender to be the Chord protocol and let the attacker evolve. On the other hand, the *Defender* column means we fixed an attacker and allowed the defender to evolve. Finally, the *Coevolution* column means we evolved both attackers and defenders against each other simultaneously.

We first consider the results of running our experiments on Topology 0, which can be found in table 4.2. When fixing the defender as the Chord protocol (*Attack* column), the values show how the different algorithms performed differently in finding attackers that were better suited at disrupting the mission. This is evident through the different levels of fitness found for defenders across all of the algorithms with the IPCA and rIPCA algorithms finding defenders with the highest fitness. When fixing the attack (*Defense* column), most algorithms converged to the Chord protocol which has a fitness of 0.250. When both attacker and defender were allowed to evolve (*Coevolution* column), the algorithms converge on Chord as the best defender with different levels of certainty.

We next consider the results from Topology 1 in table 4.3. We first note that the

Table 4.3: Network mission results for fixed attack, fixed defense and coevolution for topology 1 on the logical simulator implementation.

Algorithm	**Attack**	**Defense**	**Coevolution**
Coev	0.067 ± 0.000	0.100 ± 0.033	0.067 ± 0.031
MinMax	0.062 ± 0.000	0.100 ± 0.033	0.059 ± 0.013
MaxSolve	0.163 ± 0.002	0.111 ± 0.000	0.074 ± 0.026
IPCA	0.073 ± 0.001	0.111 ± 0.000	0.070 ± 0.003
rIPCA	0.079 ± 0.008	0.111 ± 1.387	0.068 ± 0.003

Table 4.4: Network mission results for fixed attack, fixed defense and coevolution for topology 2 on the logical simulator implementation.

Algorithm	**Attack**	**Defense**	**Coevolution**
Coev	0.173 ± 0.001	0.053 ± 0.022	0.053 ± 0.022
MinMax	0.102 ± 0.001	0.053 ± 0.022	0.057 ± 0.017
MaxSolve	0.096 ± 0.000	N/A	0.081 ± 0.027
IPCA	0.072 ± 0.000	0.062 ± 0.0	0.079 ± 0.000
rIPCA	0.073 ± 0.001	0.062 ± 0.000	0.062 ± 0.000

average fitness values of the defender populations are lower across the entire attack column than they were under Topology 0. This is not because of the performance of the algorithms, attackers, or defenders, but because of the increase in the number of nodes in the topology. The defenses' fitness function is inversely proportional to the number of hops needed to reach its goal. Thus, an increase in the size of the topology will lead to an overall decrease in fitness values for the defense population. The defense column for Topology 1 is similar to that of the defense column for Topology 0 as they all converge on Chord as the best defender solution. As for the coevolutionary results, we note again that a few of the algorithms, like IPCA and rIPCA, converged on the Chord protocol as the best solution with more certainty than the others.

Finally, Topology 2 – the largest of the topologies. Across all columns of table 4.4, we notice many of the same trends we saw from the earlier topologies. One exception that we did not know how to explain, however, was that the MaxSolve coevolutionary algorithm was unable to converge on a solution.

4.2.2 Logical to Physical Network Simulator Results

The setup for running the algorithms on the logical to physical simulator implementation was the exact same as that in the one conducted with the logical simulator implementation. The difference was the underlying Chord implementation that makes use of both the physical and logical layers, and that we did not run experiments where a defender or attacker was fixed. The results of running the algorithms on the logical to physical network simulator can be found in table 4.5. The trends in the defender fitness values across the topologies and algorithms closely resemble the trends we noted in tables 4.2, 4.3, and 4.4 in the previous experiment. The difference, however, is that upon inspecting the outputs of the algorithms, rather than algorithms converging on the Chord protocol as the best solution for the defender, it varied in Topology 0 and Topology 1 between the Chord protocol and the flooding protocol. In the largest topology, the flooding protocol was strictly found by all of the algorithms as the best defender. Given these results, we claim that increasing the complexity of the simulator to traverse the physical network between two nodes for every hop between the corresponding nodes in the virtual layer increased the number of hops the Chord protocol took to get the message to the destination. This, in turn, affected its fitness score which is why it was in direct competition with the flooding algorithm. We did observe, however, that the shortest path did not come up as a solution thus verifying that the Chord protocol and the flooding protocol are more robust in terms of withstanding attackers. Finally, another interesting observation we made when conducting this experiment was the time it took to run all the algorithms across all of the topologies for coevolution only. The amount of time it took to run just the coevolutions for all algorithms across all topologies took the same amount of time – about 13 hours – to complete as the experiments conducted on the logical simulator implementation where fixed attackers and fixed defenders were included. We hypothesize that the performance in this experiment was different than that of the logical network simulator experiments because of an increase in the number of timesteps incurred as a result of traversing both the physical and logical networks. This difference

Table 4.5: Network mission results for coevolution for all topologies on the logical to physical network simulator implementation

	Topology 0	*Topology 1*	*Topology 2*
Algorithm	**Coevolution**	**Coevolution**	**Coevolution**
Coev	0.079 ± 0.010	0.007 ± 0.001	0.005 ± 0.000
MinMax	0.053 ± 0.023	0.004 ± 0.001	0.004 ± 0.001
MaxSolve	0.061 ± 0.018	0.006 ± 0.001	0.002 ± 0.001
IPCA	0.082 ± 0.009	0.007 ± 0.000	0.005 ± 0.001
rIPCA	0.095 ± 0.027	0.008 ± 0.001	0.005 ± 0.001

in performance highlights the issues that begin to arise when the complexity of the network simulator increases.

4.2.3 Chord Plus Results

Because this experiment is still in its very early stages and because we are just beginning to hash out adversarial components such as fitness functions, and attacker and defender parameters and goals, it is hard to ground the meaning of the fitness values and solutions returned from our experiments on this software. Despite this, we did come across interesting results regarding the execution times of the experiments. In the first experiment, we ran the most basic coevolutionary algorithm with a population size of 3 and number of generations to be 2 on a very small custom topology. This very small example took about 22 minutes to run. With the same topology and the same algorithm, we slightly changed the population size to be 4 and the generations to be 10 after which the experiment took 948 minutes or 15.8 hours to complete. Both of these experiments were real time simulations of the network. This time was essentially the time it took for the logical implementation of the network simulator to run all experiments across all topologies and algorithms. In a similar way to that of the experiments run on the logical to physical simulator version, this report on performance points to the issues that arise when the complexity of the system increases.

Chapter 5

Discussion

In this chapter, we reflect on some of the key findings of the experiments and questions that may have arisen from analyzing the results of the experiments. We separate our thoughts across three different areas: the effectiveness of using coevolutionary algorithms in modeling adversarial behavior in networks, the tradeoffs of using a network simulator instead of a live system when integrating coevolutionary algorithms with the network, and thoughts on the barriers remaining to successfully use coevolutionary algorithms on the proprietary software that implements the actual Chord protocol.

5.1 Effectivness of Coevolutionary Algorithms

From the very beginning of this thesis, we hypothesized that given the correct framework to use and model to work with, that coevolutionary algorithms would be very useful in modeling adversarial behavior in networks. It is clear from the results obtained across the experiments that the coevolutionary algorithms were able to find the best average attacker solution on a network, and the best routing configurations for a network simultaneously. While this thesis focused mainly on establishing a framework that facilitates incorporating the use of these coevolutionary algorithms on a network, it is clear that the output of the algorithms can be very beneficial to an expert designing a network. By coevolving both the attacker and defender, the expert can use the information returned about the best solutions to make better and

more informed decisions when designing the network.

In addition to establishing the usefulness of coevolutionary algorithms to approach the problem of creating robust cyber defenses, it is also worthy to note that different types of coevolutionary algorithms are better suited to finding optimal solutions than others. For example, across the experiments, we typically saw IPCA and rIPCA converge on a solution with a higher confidence interval than the other algorithms such as MaxSolve, MinMax, and a generic coevolutionary algorithm. This information is useful in a future where this technology is used to create robust defenses because it shows which algorithms are the ones we should be focused on integrating and running.

5.2 Effectiveness of Network Simulator

It is clear from the fact that we were able to run any substantial experiments to show the effectiveness of coevolutionary algorithms on the problem of creating robust cyber defenses that creating a novel network simulator was the key step of this research. Without making assumptions and abstracting away all of the extraneous details that might come as a result of working with a real peer-to-peer netowrk, we would not have made much progress in the research questions we established in chapter 1. Developing and using both versions of the network simulator allowed us to control the environment and make it suitable enough for us to integrate the coevolutionary algorithms and model the coevolution of attackers and defenders on a network. As long as more work is put into slowly building the complexity of the simulator without increasing the difficulty of integrating the coevolutionary algorithms, then this is currently the best approach for modeling adversarial behavior in networks.

5.3 Thoughts on Network Emulation

Although the overall goal is to be able to model adversarial behavior in live systems, the amount of time it took to run even the simplest of experiments on the proprietary emulator shows how much work is left to be done in this area. Along with finding a

way to speed up running the algorithms on the system, there should also be much focus placed on creating a system that is modular and easy to integrate with components such as the coevolutionary algorithms. If these issues can be solved, this would be incredibly exciting as the search space for both attackers and defenders would be enormous in live systems. For example, in the proprietary emulator we use, we are able to select among countless numbers of configurations and parameters for the network thus giving it many more defense options than just three routing protocols to work with.

Chapter 6

Conclusions

In the very beginning of this thesis, we posed the two main research questions we needed to address if we were to make progress towards the overall vision of creating adaptive cyber defenses. The two questions were:

1. How can we effectively implement a peer-to-peer network simulator which interfaces with coevolutionary adversarial genetic algorithms that need to manipulate defensive configurations of the network?

2. How can we integrate the same coevolutionary genetic algorithms with a live peer-to-peer simulator with far more details and complexities and set out an example of evaluation?

To address these questions, we designed and implemented a network simulator that implements the Chord protocol and integrated this simulator with several coevolutionary algorithms to see if this was an effective way to model the adversarial behavior between a cyber attack and cyber defender. Second, we also incorporated one of the coevolutionary algorithms into a novel proprietary software implementing an enhanced version of the Chord protocol and obtained initial results. Through our experiments, we were able to determine that coevolutionary algorithms are indeed effective in determining optimal configurations for a network and can be used to make better decisions when designing a network, but deteriorate in performance as the complexity of the system they are used in increases.

Although this thesis takes us one step closer to achieving the vision of creating robust cyber defenses in order to better protect networks from adaptive cyber attacks, there is still a great deal of work to be done. Future work that could stem from this thesis includes: improving the complexity and realism of the network simulator while maintaining a level of modularity, developing similar network simulators for different peer-to-peer protocols, and reducing the performance bottlenecks associated with modeling adversarial behavior on live systems.

Bibliography

[1] Jesimar da Silva Arantes, Márcio da Silva Arantes, Claudio Fabiano Motta Toledo, and Brian Charles Williams. A multi-population genetic algorithm for UAV path re-planning under critical situation. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 486–493, 2015.

[2] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[3] D. Garcia, A. Erb Lugo, E. Hemberg, and U. O'Reilly. Investigating coevolutionary archive based genetic algorithms on cyber defense networks. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation*, GECCO '17. ACM, 2017.

[4] Fariba Haddadi and A Nur Zincir-Heywood. Botnet detection system analysis on the effect of botnet evolution and feature representation. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, pages 893–900. ACM, 2015.

[5] Erik Hemberg, Jacob Rosen, Geoff Warner, Sanith Wijesinghe, and Una-May O'Reilly. Detecting tax evasion: a co-evolutionary approach. *Artificial Intelligence and Law*, 24(2):149–182, 2016.

[6] P. Hingston and M. Preuss. Red teaming with coevolution. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1155–1163, June 2011.

[7] Edwin D de Jong. A monotonic archive for pareto-coevolution. *Evolutionary Computation*, 15(1):61–93, 2007.

[8] Aleksandar Kuzmanovic and Edward W Knightly. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86. ACM, 2003.

[9] George Rush, Daniel R Tauritz, and Alexander D Kent. Coevolutionary agent-based network defense lightweight event system (candles). In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, pages 859–866. ACM, 2015.

[10] Travis Service and Daniel Tauritz. Increasing infrastructure resilience through competitive coevolution. *New Mathematics and Natural Computation*, 5(02):441–457, 2009.

[11] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[12] Stuart Wagner, Eric Van Den Berg, Jim Giacopelli, Andrei Ghetie, Jim Burns, Miriam Tauil, Soumya Sen, Michael Wang, Mung Chiang, Tian Lan, et al. Autonomous, collaborative control for resilient cyber defense (accord). In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on*, pages 39–46. IEEE, 2012.

[13] Michael L Winterrose and Kevin M Carter. Strategic evolution of adversaries against temporal platform diversity active cyber defenses. In *Proceedings of the 2014 Symposium on Agent Directed Simulation*, page 9. Society for Computer Simulation International, 2014.