DEFENSE AGAINST NETWORK ATTACKS USING GAME THEORY

BY

TANMAY SANJAY KHIRWADKAR

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Advisor:

Professor David Nicol

Abstract

As Internet has become ubiquitous, the risk posed by network attacks has greatly increased. Network attacks have been used to achieve a wide gamut of objectives ranging from overloading a website to accessing classified data. Effective defense against such attacks is a critical research area. In this thesis, we demonstrate how game theory can be used to devise effective defense systems.

We utilize game theory for defense systems in two scenarios in this thesis. The first scenario is that of the attacker carrying out a Distributed Denial of Service (DDoS) attack. The second scenario involves the attacker possessing the ability to carry out a number of different attacks such as Denial of Service (DoS), Dictionary attacks and Portscans.

An important restriction imposed in repeated complete-information games is that each player has complete knowledge of the adversary's payoffs. This assumption is unrealistic when the adversaries are the defense system and the attacker. We employ a Fictitious-Play approach in order to remove this restriction.

Acknowledgement

I would like to thank my advisor, Professor David Nicol, without whose invaluable guidance this work would not have been possible.

Table of Contents

Chapter 1: Introduction **1**

Chapter 2: Background **5**
2.1 Network Attacks 6
2.1.1 Denial of Service Attacks 6
2.1.1.1 SYN-Flood 7
2.1.1.2 Smurf 8
2.1.1.3 Defense 9
2.1.2 Remote Password Cracking 12
2.1.2.1 Dictionary Attack 13
2.1.2.2 Defense 13
2.1.3 Port-Scanning 14
2.1.3.1 SYN-Scanning 14
2.1.3.2 Defense 15
2.2 Game Theory 15
2.2.1 Static Games 16
2.2.2 Fictitious Play Process 17

Chapter 3: Defense Against DDoS Attacks **19**
3.1 Introduction 19
3.2 Implementation Details 22
3.2.1 PRIME Network Simulator 22
3.2.2 Network Topology 22
3.2.3 Game Formulation 26
3.2.4 Flow-based Simulation vs. Packet-based Simulation . . 28
3.3 Simulation Results 31
3.3.1 Packet-based Simulation Results 31
3.3.2 Flow-based Simulation Results 34

Chapter 4: Defense Against Multiple Attacks **39**
4.1 Introduction 39

4.2 Emulation 41
4.3 Attack Implementation 42
4.3.1 SYN-Flood 43
4.3.2 Smurf 44
4.3.3 Dictionary 45
4.3.4 Portscan 45
4.4 Defense Implementation 46
4.4.1 SYN-Flood Defense 46
4.4.2 Smurf Defense 46
4.4.3 Dictionary Defense 47
4.4.4 Portscan Defense 47
4.5 Game Formulation 49
4.5.1 Action Space 50
4.5.2 Payoffs 50
4.5.3 Start State 52
4.5.4 Optimal Strategies 53
4.6 Fictitious Play Simulation 55
4.6.1 Empirical Frequency Measurement 55
4.6.2 Results 57

Chapter 5: Conclusion **60**

Bibliography **63**

Chapter 1

Introduction

In today's world, use of internet and mobile applications is growing rapidly. A wide variety of information is shared and disseminated over networks to geographically distant resources and consumers. As networks and information systems become complex, the security and privacy problems faced by such systems have also evolved.

A large number of defense mechanisms have been proposed against network attacks. Given the sophistication and variety of today's attacks, a defense system may need to employ more than one defense mechanism to counter attacks. Even when employing a single defense mechanism, there are a number of variables involved. For example, a common defense against Denial of Service (DoS) attacks is identifying attack packets and filtering them. This defense mechanism has a number of variable parameters such as rate and duration of filtering the packets. The 'optimal' value for these parameters will depend upon the objectives of the defense system. Similarly, the combination of de-

fense mechanisms to be employed will depend upon the costs and benefits of the mechanism to the system.

An attacker also needs to choose the right combination of attacks suited to achieve his goals. For example, given two attacks - one which is fast but easy to detect and expensive to implement and the other which is slow but hard to detect and cheap to implement; the optimal choice of attack will depend upon the objective and resources of the attacker as well as the nature of defense employed by the victim against such an attack.

In light of these arguments, a network attack and a defense against it may be viewed as a *game* played between entities with conflicting objectives. Game Theory provides an excellent mathematical framework for analyzing such situations.

Some research has been done in this area. In [36], a stochastic game is used to compute probabilities of an expected attacker behavior and these probabilities are used in a transition matrix model to assess security in an interconnected system. In [3], a two-person zero-sum Markov game is proposed to capture the interactions between malicious attackers and an IDS. They also study limited information cases where players optimize their strategies offline or online depending on the type of information available, using methods based on Markov decision process and Q-learning. In [22] , a dynamic Bayesian game approach is used to analyze the interactions between pairs of attacking and defending nodes in wireless ad hoc networks where the defender updates its belief on his opponent. The authors show that a Bayesian hybrid detection switching between lightweight and heavyweight monitoring leads to detection

energy efficiency for the defender.

In our work, we use an incomplete-information repeated game to model interaction between an attacker and a defense system in two scenarios. The first scenario involves an attacker carrying out a Distributed Denial of Service (DDoS) attack. The second scenario involves an attacker possessing the ability to carry out a number of different attacks such as Denial of Service (DoS), Dictionary attacks and Portscans. We utilize game-theory to devise optimal strategies for the attacker and the defense system. Using network simulations in SSFNet [20] and emulations in Virtual Network User Mode Linux (VNUML) [7], we demonstrate the practical application of this approach.

Both [36] and [22] use a game model in which the payoff function of the adversary is known to each player. [3] addresses games in which players have limited information. However, the naive Q-learning method used by the authors in the case of unknown adversary payoff functions is not effective. In our work, we use a repeated game model based on the fictitious play (FP) process to overcome this limitation. In a FP process, each player estimates the payoff function of the adversary by observing all the actions up to present. At each stage, the player updates this estimate and plays the pure strategy that is the best response to the estimate. It has been shown that, for many classes of games, such a FP process will finally render both players playing the Nash equilibrium.

The rest of this report is organized as follows. Chapter 2 gives background information about network attacks and defenses and a brief introduction to Game Theory. In Chapter 3, we present a game-theoretic defense against

DDoS attacks. In Chapter 4, we present a game-theoretic defense against an attacker carrying out multiple types of attacks. Finally, we end with concluding remarks in Chapter 5.

Chapter 2

Background

The Morris worm in 1988 was one of the first known computer worms distributed via the Internet. While the damage caused by the worm was significant, the intent behind it was not malicious according to its creator. Today, most network attacks are malicious. With the growth of internet over the past decade, the number of attacks has increased rapidly. According to CERT, the number of reported Internet security incidents have jumped from 6 in 1988 to 137,529 in 2003 [33]. In fact, due to the excessive number of security incidents, CERT has decided not to publish the number of reported incidents since 2004. We now describe some well-known network attacks and proposed defenses against them.

2.1 Network Attacks

2.1.1 Denial of Service Attacks

A Denial-of-Service attack (DoS) is an attempt to make a computer resource unavailable to its intended users. DoS attacks generally send large volumes of packets that use up a significant proportion of the available network bandwidth in order to make resources unavailable. Hence, DoS attacks are also called bandwidth attacks. The aim of a bandwidth attack is to consume critical resources in a network service. Some of the resources that may be targeted by such attacks are CPU capacity in a server, stack space in network protocol software, or Internet link capacity. A Distributed Denial-of-Service attack (DDoS) is a DoS attack in which the attacker uses more a than one host, router or other network device to generate the attack traffic. An important feature of DoS attacks is that volume of traffic rather than the content of traffic is used to make the targeted resource unavailable.

A typical DDoS attack has two phases -

1. Exploit vulnerabilities in systems on the network to install attack tools in such systems

2. Send command messages to compromised systems ("zombies") through a command-channel to carry out bandwidth attack against a target

There are two main effects of a bandwidth attack. First, it consumes the resources of the target which is typically a server. A server has a finite capacity and resources to process incoming requests (packets). A bandwidth attack

uses up this capacity through the volume of attack traffic. As a result, the server is unable to process requests from legitimate users and becomes unavailable for such users. The second effect is consumption of network bandwidth. Consumption of network bandwidth affects not only the legitimate traffic to the targeted service but also legitimate traffic to other services sharing the affected network link.

Two classic bandwidth attacks are the SYN-Flood and Smurf attacks which exploit weaknesses in internet protocols.

2.1.1.1 SYN-Flood

As described in [9], SYN-Flood attacks first came to light in 1996, when the description and source code of the attack was published by the magazines Phrack and 2600. The basis of the SYN flooding attack lies in the design of the 3-way handshake that establishes a TCP connection. In a three-way handshake -

1. The client requests a connection by sending a SYN (synchronize) message to the server.

2. The server acknowledges this request by sending SYN-ACK back to the client.

3. The client responds with an ACK to complete the connection.

SYN-Flood attacks are of two types. A malicious agent can skip sending the ACK message; or by spoofing the source ip address in the SYN, it can make the server send the SYN-ACK to the falsified ip address and thus never receive

the ACK. In both cases, the server never receives the final ACK message. Each such half-open connection consumes resources on the server and may lead to exhaustion of host's kernel memory. In order to avoid this memory exhaustion, operating systems generally associate a "backlog" parameter with a listening socket that sets a cap on the number of connections simultaneously in the SYN-RECEIVED state. Although this action protects a host's available memory resource from attack, the backlog itself represents another (smaller) resource vulnerable to attack. With no room left in the backlog, it is impossible to service new connection requests until some connections are removed from the SYN-RECEIVED state.

2.1.1.2 Smurf

The Internet Control Message Protocol (ICMP) is based on the IP protocol and is used to diagnose network status. A Smurf attack is a DoS attack that uses ICMP packets. This attack relies on a perpetrator sending a large amount of ICMP echo request (ping) traffic to ip broadcast addresses, all of which have a spoofed source ip address of the intended victim. If the routing device delivering traffic to those broadcast addresses delivers the ip broadcast to all hosts, most hosts on that ip network will take the ICMP echo request and reply to it with an echo reply, multiplying the traffic by the number of hosts responding. On a multi-access broadcast network, hundreds of machines might reply to each packet.

2.1.1.3 Defense

Defense mechanisms proposed in research literature against DoS (and DDoS) attacks can be divided into three main categories -

1. Attack Prevention
2. Attack Detection
3. Attack Reaction

Attack Prevention Attack Prevention aims to preempt attacks before they cause damage. This approach is effective against DoS attacks in which the source address of attack traffic is spoofed to hide the real source of the attack traffic and exploit protocol vulnerabilities. The main exponents of this approach is Ingress/Egress Filtering [10], Router-based Packet Filtering (RPF) [30] and Source Address Validity Enforcement (SAVE) [17].

Ingress filtering involves filtering the traffic coming into a local network, and egress filtering involves filtering the traffic leaving a local network. The purpose of ingress/egress filtering is to only allow traffic to enter or leave the network if its source addresses are within the expected ip address range in the network. Thus, as a result of deploying ingress/egress filtering, spoofed ip packets with source ip address not within the network are dropped, thereby mitigating the effect of DoS attacks.

RPF extends ingress filtering to the core of the Internet. It is based on the principle that for each link in the core of the Internet, there is only a limited set of source addresses from which traffic on the link could have originated.

In the event that an unexpected source address appears in an ip packet on a link, we can infer that the source address has been spoofed, and hence filter the packet.

SAVE uses a protocol that can provide routers with information needed for source address validation. SAVE messages propagate valid source address information from source location to all destination, allowing each router along the way to build an incoming table that associates each incoming interface of the router with a set of valid source address blocks.

The techniques described above involve changes in network infrastructure and protocols. Hence, unless policies or regulations are implemented for their enforcement, it is difficult to deploy such techniques.

Attack Detection There are two main categories of DoS attack detection techniques - *DoS-attack-specific* detection and anomaly detection.

DoS-attack-specific detection utilizes characteristics of DoS attack traffic. Since DoS traffic is generated by the attacker, it does not typically follow traffic control protocols. There is an imbalance in the traffic between source and victim since the victim is not able to handle all incoming packets. This is not the case for normal traffic. MULTOPS [12] is based on the assumption that packet rates between two hosts are proportional during normal operation. It monitors traffic rates in up and down links to detect disproportional traffic between hosts in order identify DoS attacks. TOPS [2] uses a similar approach but is more memory efficient on account of use of hashing scheme with a small set of field length lookup tables. Other methods such as [39] define a statistical

model of normal traffic and then identify traffic which does not match this model to be attack traffic.

Anomaly detection builds a model of normal traffic using training data. If the monitored traffic is statistically different from the model, then it can be inferred that a DoS attack is in action. The first real-time intrusion detection model was proposed in [8]. It detected attacks by monitoring a system's audit records for abnormal patterns of system usage.

The main drawback of using attack detection techniques mentioned above is that depend heavily upon a traffic model which may not be universally applicable. Building a traffic model and making online statistical comparison between normal and observed traffic is also time-consuming and costly.

Attack Reaction The main aim of DoS attacks is to damage the target as much as possible. Attackers typically do not disguise the attack since the target will be aware of the attack damage eventually. The attack detection techniques mentioned attempt to detect an ongoing attack in the minimal possible time. In order to minimize the damage caused by DoS attacks, a reaction scheme must be employed after an attack has been detected.

DoS attacks not only affect the end-host victim but also congest the intermediate links between the source and the victim. Attack reaction will be most effective if the attack traffic is filtered as close to the source as possible. Attack reaction techniques can be classified into host-based reaction which takes place only at the end host and network-based reaction which takes place at intermediate routers (and optionally end-hosts).

An example of network-based reaction is [23]. It uses an online scheme in which intermediate routers learn a congestion signature based on the victim's ip address and the volume of traffic directed towards that ip address. Once a signature has been identified local congestion control is filter attack traffic. In addition, a *Pushback* mechanism is used to request upstream adjacent routers to rate-limit traffic matching a specified signature. [31] uses a *Selective Pushback* mechanism that sends pushback messages to the routers closest to the attack sources directly by analyzing the traffic distribution change of all upstream routers at the target.

The techniques mentioned above, though effective, require the co-operation of routers for their implementation. In many cases, a victim may not have access to such routers. In such a scenario, defense mechanism must be implemented on the end-host alone. While such techniques cannot completely stop an attack, they can mitigate the damage caused. An example of such a technique is SYN-cookies [5] using which a host does not need to keep track of half-open connection states thereby mitigating the effects of a SYN-Flood. Another technique is using system and network interface logs on the host to identify ip address from which malicious traffic is originating and then filtering them using tools such as *iptables/netfilter* [1]. Malicious traffic can also be identified using a history maintained by the host [32].

2.1.2 Remote Password Cracking

In a Remote Password Cracking attack an attacker tries to gain unauthorized access to some machine by making repeated guesses at possible usernames

and passwords. Password guessing can be done remotely with many services; telnet, ftp, pop, rlogin, and imap are the most prominent services that support authentication using usernames and passwords. Dictionary attack is one such type of attack.

2.1.2.1 Dictionary Attack

A Dictionary attack uses a targeted technique of successively trying all the words in an exhaustive list called a dictionary which is a pre-arranged list of values. In contrast with a brute force attack, where a large proportion key space is searched systematically, a dictionary attack tries only those possibilities which are most likely to succeed, typically derived from a list of words for example a dictionary or a bible etc. Dictionary attacks succeed because many people have a tendency to choose passwords which are short (7 characters or fewer), single words found in dictionaries or simple, easily-predicted variations on words, such as appending a digit.

2.1.2.2 Defense

The best defense against a password cracking attack is to choose strong passwords. However, since many users fail to do that, some other common defense mechanisms against online Dictionary attacks are -

1. **Delayed Response**: Given a login-name/password pair the server provides a slightly delayed yes/no answer (say not faster than one answer per second). This should prevent an attacker from checking sufficiently many passwords in a reasonable time.

2. **Account Locking**: Accounts are locked a few unsuccessful login attempts (for example, an account is locked for an hour after five unsuccessful attempts.) Like the previous measure, this measure is designed to prevent attackers from checking sufficiently many passwords in a reasonable time.

3. **Ip - Blocking:** With account locking, accounts belonging to legitimate users can be locked if some other user makes unsuccessful attempts to login. One way to prevent this is to keep track of ip addresses from which login attempts have been made and block an ip in case of a specific number of unsuccessful login attempts.

2.1.3 Port-Scanning

A Portscan attack sends client requests to a range of server port addresses on a host, with the goal of finding an active port and exploiting a known vulnerability of that service. There are many varieties of portscanning. We consider SYN scanning in particular which is also known as "half-open scanning".

2.1.3.1 SYN-Scanning

In SYN scanning, the hostile client attempts to set up a TCP/IP connection with a server at every possible port. This is done by sending a SYN (synchronization) packet, as if to initiate a three-way handshake, to every port on the server. If the server responds with a SYN/ACK (synchronization acknowledged) packet from a particular port, it means the port is open. If the server

responds with an RST (reset) packet from a particular port, it indicates that the port is closed and cannot be exploited.

2.1.3.2 Defense

Tools such Port Scan Attack Detector (PSAD) [34] known signatures of portscanning traffic to scans. PSAD requires the iptables/netfilter to log all ip traffic.

Another simpler mechanism can be used to detect SYN scanning. Since an attacker will typically use SYN scanning on all ports to carry out an attack, the defense system can monitor traffic on an arbitrary port on which no service is running. A legitimate client typically won't send a SYN packet to this port since no service on the system uses this port. Hence, if SYN packets are being sent to this port, it means that a SYN-scan is in progress.

2.2 Game Theory

In the earlier section, we have described some common network attacks and defenses against them. In our work, we propose to use game theory, specifically a fictitious play model of game theory to devise optimal strategies for an attacker and a defense system, given a choice of attacks and defenses against them. We now give an overview of static games and fictitious play, where player P_1 has m and player P_2 has n possible actions (pure strategies) [27, 28, 37, 38].

2.2.1 Static Games

We first introduce a one-shot nonzero-sum version of the games, which we will refer to as static games. In equations written for the generic player P_i, $i = 1, 2$, we use k to denote m or n. We denote by $p_1 \in \triangle(m)$ and $p_2 \in \triangle(n)$ a pair of mixed strategies for P_1 and P_2, respectively, where $\triangle(k)$ is the simplex in $\mathbb{R}^k$ ($\mathbb{R}$ is the set of real numbers), i.e.,

$$\triangle(k) = \left\{ s \in \mathbb{R}^k \mid s_j \geq 0,\ j = 0, 1, \ldots, k,\ \sum_{j=1}^{k} s_j = 1 \right\} \tag{2.1}$$

For a static game, player i selects an integer action v_i according to the mixed strategy p_i. The (instant) payoff for player P_i is $v_i^T M_i v_{-i}$, where we use $v_i^{(j)}$, $j = 1, \ldots, k$, to indicate the j^{th} vertex of the simplex $\triangle(k)$. (For example, when $k = 2$, $v_i^{(1)} = [1\ 0]^T$ for the first action, and $v_i^{(2)} = [0\ 1]^T$ for the second action).[1] For a pair of mixed strategies (p_1, p_2), the utility functions are given by the expected payoffs:

$$\begin{aligned} U_i(p_i, p_{-i}) &= E[v_i^T M_i v_{-i}] \\ &= p_i^T M_i p_{-i} \end{aligned} \tag{2.2}$$

where M_i is the payoff matrix of P_i, $i = 1, 2$ (Note that M_1 is of dimension $m \times n$ and M_2 $n \times m$.) This standard formulation of the payoff functions leads to a FP process that will be referred to as classical FP.

[1] As standard in the game theory literature, the index $-i$ is used to indicate those of other players, or the opponent in this case.

Now, the best response mappings $\beta_1 : \triangle(n) \rightarrow \left\{v_1^{(1)}, v_1^{(2)}, \ldots, v_1^{(m)}\right\}$ and $\beta_2 : \triangle(m) \rightarrow \left\{v_2^{(1)}, v_2^{(2)}, \ldots, v_2^{(n)}\right\}$ are defined as:

$$\beta_i(p_{-i}) = \arg \max_{v_i \in \left\{v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(k)}\right\}} U_i(v_i, p_{-i}) \tag{2.3}$$

Note that given p_{-i}, the best response mapping can be set-valued, i.e., there may be multiple vertices of the simplex $\triangle(k)$ that yield the maximum value of the payoff function. Finally, a (mixed strategy) Nash equilibrium is defined to be a pair $(p_1, p_2) \in \triangle(m) \times \triangle(n)$ such that for all $p_1 \in \triangle(m)$ and $p_2 \in \triangle(n)$:

$$U_i(p_i, p_{-i}^*) \leq U_i(p_i^*, p_{-i}^*) \tag{2.4}$$

In a static game, as both players act simultaneously, each player cannot observe the action of the other. Their choice of actions (or probabilities of actions – mixed strategies) is based on their knowledge of both payoff matrices.

2.2.2 Fictitious Play Process

From the static game described in Section 2.2.1, we define the (discrete-time) FP process as follows. The game is now repeated at times $\tau \in 0, 1, 2, \ldots$. The empirical frequency $p_i(\tau)$ of player P_i is given by:

$$p_i(\tau + 1) = \frac{1}{\tau + 1} \sum_{j=0}^{\tau} v_i(j) \tag{2.5}$$

Using induction, we can prove the following recursive relation:

$$p_i(\tau+1) = \frac{\tau}{\tau+1}p(\tau) + \frac{1}{\tau+1}v_i(\tau) \tag{2.6}$$

Player i , $i = 1, 2$, then employs the algorithm listed in Algorithm 2.1.

Algorithm 2.1 Fictitious Play Algorithm

1: Given payoff matrix M_i
2: **for** $\tau \in \{0, 1, 2, \ldots\}$ **do**
3: Update the empirical frequency of the opponent $p_{-i}(\tau+1)$ using Eq. 2.6
4: Pick the optimal pure strategy(s) using Eq. 2.3
5: If there are multiple optimal strategies, randomize over optimal strategies with equal probability
6: **end for**

Thus, in terms of information, there are two main features that distinguish a FP process from the corresponding static game (the static game with the same payoff formulation). First, each player can make decisions without necessarily knowing the other's payoff matrix. Second, each player has to be able to observe the other's actions. It is however worth noting that such a mechanism does not guarantee convergence to a Nash equilibrium. For two-player zero-sum FP, the convergence proof was obtained for arbitrary numbers of actions for each player ($m \times n$) [35]. For nonzero-sum games, the proofs for two-player FP have been found for the case where one player is restricted to 2 actions (See [4] for classical FP and [37] for stochastic FP). Nevertheless, there are counter examples (e.g., for 3×3 games) where FP does not converge to the mixed strategy NE [38]. Several techniques that can be used to enhance convergence are discussed in [38, 26].

Chapter 3

Defense Against DDoS Attacks [1]

3.1 Introduction

As described in Section 2.1.1, a Distributed Denial-of-Service attack (DDoS attack) is an attack launched from multiple computers in a network to flood the resources of a targeted system, and thus making it less accessible to the intended users. The computers launching attacks are called zombies, which could be regular hosts that have been compromised by the attacker. In this work, we utilize the pushback defense, a mechanism first proposed in [23] and combine it with game theory.

As described in [13, 23], pushback is a mechanism that allows routers in a network to cooperate in *aggregate-based congestion control* (ACC). An aggregate is defined to be a collection of packets that share a common property or parameter, such as ICMP ECHO packets or packets with the same desti-

[1] The work described in this chapter has been published as [15]

nation ip address. The properties or parameters used to identify an aggregate are called *attack signatures*. Based on aggregates, traffic and packets are divided into three different categories: "bad", "poor", and "good". Bad traffic is that generated by the attackers. Poor traffic is from legitimate users but shares the same attack signatures. Finally, good traffic does not match the attack signatures but may suffer from the congestion. In local ACC, an individual router identifies the aggregates that causes the congestion and tries to cut down the throughput of these aggregates. In pushback, a router can request adjacent upstream routers to rate-limit some aggregates. This way, the system can save the bandwidth that would otherwise be wasted if packets in these aggregates were dropped downstream. Furthermore, if the DDoS attack traffic comes from a few upstream links, pushback helps protect poor traffic from congestion due to attack traffic.

In this chapter, we will use "Attacker" to refer to the DDoS attacker and all the zombies under its control, and "System" to refer to all the routers taking part in the pushback mechanism. When the Attacker launches DDoS attacks, it has at its disposal a number of strategies to choose from. Among these are the set of zombies, the set of targeted computers, and the attack protocols and traffic patterns. Similarly, the System can also change the pushback parameters such as the congestion checking time, the target drop rate, and the aggregate pattern. For each pair of strategies of the Attacker and the System, the payoffs for each of them can be formulated based on the bandwidth occupied by Attacker, the bandwidth used by the legitimate users, and the costs of attacking and defending. It thus can be seen that there is a game situation

between the Attacker and the System, where each player tries to maximize its own payoff against all the possible strategies of the opponent.

In [21], DDoS attacks are modeled as a Bayesian game among the Attacker, the System, and legitimate users. With such a game formulation, in order to compute a Nash equilibrium pure or mixed strategy, each player has to have full knowledge of payoff. The paper also mentions a repeated mechanism where at each step, each player makes the best response to current strategies of other players. Although this mechanism allows each player to proceed without necessarily knowing others's payoff matrices, it works well only when the game has a pure strategy Nash equilibrium.

We examine a repeated game model based on the fictitious play (FP) process for pushback defense. In a FP process, each player observes all the actions up to present and makes estimate of the mixed strategy of the opponent's actions. At each stage, the player updates this estimate and plays the pure strategy that is the best response to the estimate. It can be seen that in a FP process, if one plays a fixed strategy (either of the pure or mixed type), the opponent's strategy will converge to the best response to this fixed strategy. Furthermore, it has been shown that, for many classes of games, such a FP process will finally render both players playing the Nash equilibrium.

3.2 Implementation Details

3.2.1 PRIME Network Simulator

In this work we use the PRIME Network Simulator/Emulator. PRIME stands for Parallel Real-time Immersive Modeling Environment [19, 20, 18]. PRIME is intended to simulate large-scale computer networks with thousands to millions of network entities. PRIME has two main components: PRIME SSF (Scalable Simulation Framework) and PRIME SSFNet. While SSF is the kernel that supports parallel and real-time simulation, PRIME SSFNet is the upper layer providing network simulation functions.

3.2.2 Network Topology

The network topologies used in the simulations are shown in Fig. 3.1 (Network I) and Fig. 3.2 (Network II). The differences between these two networks are the bandwidths of the links: Network II features a much larger scale and up-to-date network with $OC{-}3$ and $OC{-}48$ links connecting backbone routers. We implement packet-based simulation for Network I and flow-based simulation for Network II (described in Section 3.2.4). Of the 64 hosts, U_1 to U_{64}, there are 8 zombies and 56 legitimate users. Both zombies and users send packets to servers S_1 and S_2. The routers $R_{i.j}$, $i = 0, 1, 2, 3$, are organized in a hierarchical manner where the subscript i denotes the level, and the subscript j denotes the router in a level. On the Attacker's side, a central controller controls all the zombies in the network using control messages. Similarly, on the System's side, a master router controls the (slave) routers taking part in

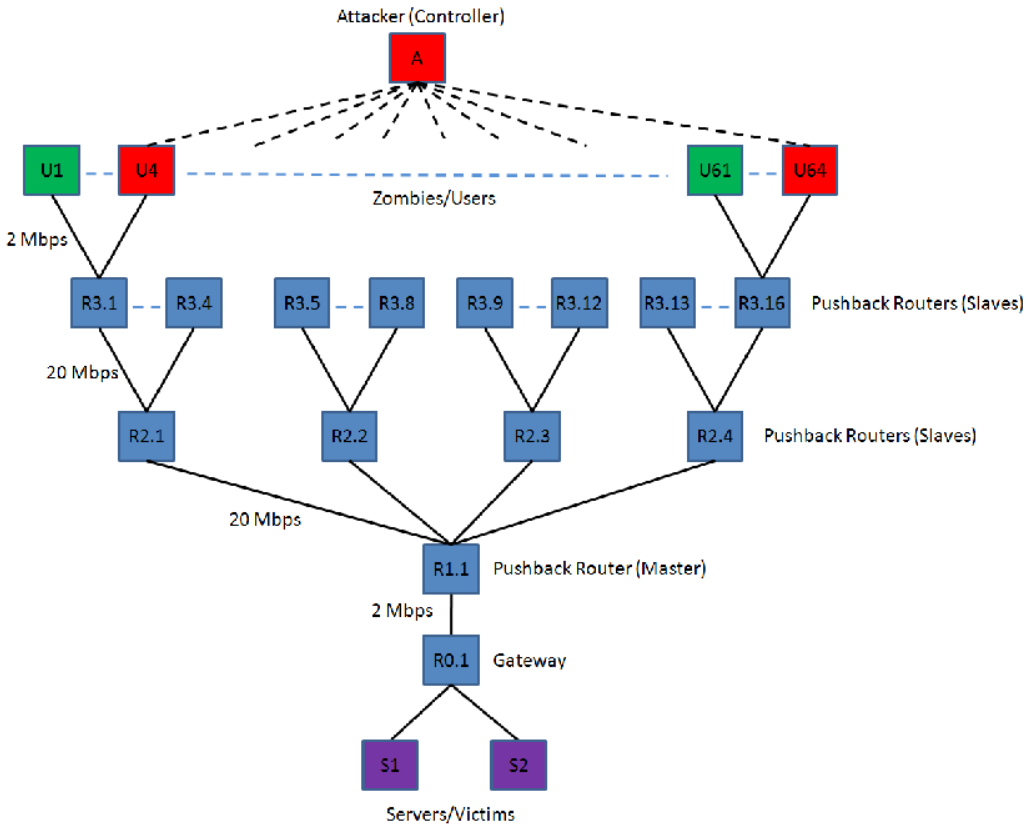

Figure 3.1: Network I with packet-based simulation

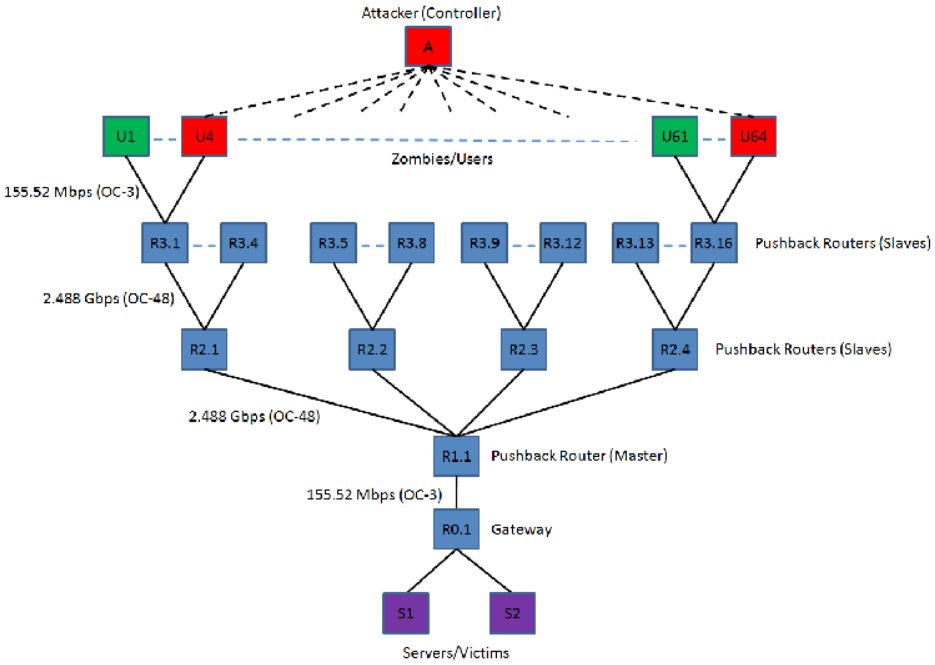

Figure 3.2: Network II with OC−3 and OC−48 links used for flow-based simulation

the pushback mechanism with pushback control messages.

Each router employs a version of the ip protocol with modifications for enforcing pushback [23, 13]. Every router checks for congestion after each specified time interval which we refer to as the $Congestion\_Checking\_Interval$. A router is considered to be in congestion if $Incoming\_Data\_Rate > (1 + Target\_Drop\_Rate) \times Outgoing\_Bandwidth$. Here, $Target\_Drop\_Rate$ is the acceptable rate of dropping packets for the router. If a router detects congestion, it looks through the log of dropped packets that it maintains to identify an attack signature. Since the source ip-address of a packet can be spoofed by the Attacker, we only use the destination ip-address as the attack signature. Thus, the router identifies the most frequently occurring destination ip-address in the dropped packets log as the signature. For the sake of efficiency, the log is of a fixed size and new log records overwrite older ones if the log is full. In subsequent checks for congestion, if the router detects that the incoming traffic not matching the signature is still greater than $(1 + Target\_Drop\_Rate) \times Outgoing\_Bandwidth$ then each time it adds the next-most frequently occurring destination ip-address in the log to the current signature. Each signature has a timestamp which is updated every time the router detects congestion. A router also sends the identified signatures to its immediately upstream routers and uses signatures received from downstream routers as attack signatures. Traffic through the router which matches the signature (*Signature Traffic*) is filtered out. The maximum signature traffic allowed to pass through the router is $Outgoing\_Bandwidth \times (1 + Target\_Drop\_Rate) - Non\_Signature\_Traffic$. A router also periodically

Networks	A1	A2	A3	A4	A5	Users' Datarates
Network I	91.73 Mbps	55.04 Mbps	27.52 Mbps	2.75 Mbps	0.275 Mbps	1.93 Mbps
Network II	13.76 Gbps	5.504 Gbps	2.752 Gbps	0.275 Gbps	30.58 Mbps	124.42 Mbps

Table 3.1: Attacker's Actions (Datarates generated by all 8 zombies) and Collective Users' Datarates

checks if any of the attack signatures has expired after a specified time interval (which we refer to as $Refresh\_Interval$). Routers periodically send update messages for signatures to upstream routers. Routers use update messages from downstream routers to update the timestamp of the signatures received from downstream routers.

3.2.3 Game Formulation

The Attacker's pure strategies are given by $A_{att} = A_1, \ldots, A_5$, where A_i, $i = 1, \ldots, 5$ are the collective attack datarates (generated by all 8 zombies). The Attacker's actions are shown in Table 3.1. The System consists of all the routers taking part in pushback defense, $R_1, \ldots, R_8$. The pushback behavior of a router is represented by three parameters: $Congestion\_Checking\_Interval$, $Refresh\_Interval$, and $Target\_Drop\_Rate$. The action space of the System, $A_{sys} = S_1, \ldots, S_6$, is the same for both Networks I and II, and is specified in .Table 3.2

For each pair (A_i, S_j), $i = 1, \ldots, 5$, $j = 1, \ldots, 6$, the payoff of the Attacker is given by:

Actions	Congestion Checking Interval (sec)	Refresh Interval (sec)	Target Drop Rate
S1	2	5	0.05
S2	2	10	0.05
S3	4	5	0.05
S4	2	5	0.03
S5	6	10	0.05
S6	2	5	0.07

Table 3.2: System's Actions

$$U_{att} = \alpha \frac{B_{ao}}{B_N} + (1 - \alpha) \left(1 - \frac{\sum_{l=1}^{L} B_{lo}^{(l)}}{\sum_{l=1}^{L} B_{lw}^{(l)}} \right) \tag{3.1}$$

where $B_{lo}^{(l)}$ is the bandwidth occupied by the legitimate user l, and $B_{lw}^{(l)}$ is the bandwidth required by the legitimate user l, $l = 1, \ldots, L$, where L is the number of legitimate users (56 in our simulations), B_{ao} is the bandwidth occupied by the Attacker, and B_N is the bandwidth capacity ($B_N = 2$ Mbps for Network I and $B_N = 155.52$ Mbps for Network II). $\alpha \in [0, 1]$ is used to balance between the damage the Attacker does to the System and the damage it causes to the legitimate users; α is chosen to be 0.2 throughout our simulations. The payoff of the System is given by:

$$U_{sys} = \omega \frac{\sum_{l=1}^{L} B_{lo}^{(l)}}{\sum_{l=1}^{L} B_{lw}^{(l)}} + (1 - \omega) \left(1 - \frac{B_{ao}}{B_N} \right) \tag{3.2}$$

where $\omega \in [0, 1]$ is used to balance between the utility the System can provide for the legitimate users and the pushback it applies against the Attacker; ω is chosen to be 0.8 throughout our simulations. The costs of attacking and defending can also be included in the payoff functions. For the Attacker, the

action to be taken is determined by the controller and sent to the zombies. The zombies then adjust their datarates and pick their victims accordingly. Similarly, for the System, the action to be taken is determined by the master router and sent to the slave routers. The slave routers then adjust their pushback parameters accordingly. Our simulations consists of two steps: payoff measurement and fictitious play. In the first step, the System and the Attacker are forced to take each pair of actions. The attack traffic, good traffic, and poor traffic at router $R_{0.0}$ are then measured. These measurements are used to calculate the payoffs for the Attacker and the System using Eq. 3.1 and Eq. 3.2 respectively. In the second step, both the System and the Attacker use a fixed time interval as a "time step", during which the action taken by the opponent is identified. At the end of each time interval, both players choose the next action to be taken which is the best response to the empirical frequencies of the opponent's actions (using Algorithm 2.1 with the payoff matrices obtained from Step 1). The time step is chosen to be 50s, which allows enough time for the pushback mechanism to stabilize.

3.2.4 Flow-based Simulation vs. Packet-based Simulation

In packet-based simulation (Network topology I in Fig. 3.1), the Attacker's traffic consists of fixed length ip packets generated at a constant rate by the zombies. Users' traffic consists of fixed length ip packets with inter-packet times being exponentially distributed. For determining the parameter of the exponential distribution, we set the average user datarate to be the bandwidth

of the router $R_{0.0}$ (Fig. 3.1) divided by the number of hosts in the network. The rationale behind this is that if all users send out data with this rate, there should be no congestion in the network. The datarates generated by the each zombie ranges from around 300 to 1 times the legitimate user datarate.

We could run simulations with network bandwidths in the order of 20Mbps using packet-based simulation with PRIME SSFNet. Simulations for networks with larger bandwidths require significantly long times. Thus for Gigabit networks (Network II, Fig. 3.2), we adopt the flow-based simulation approach [29]. In this approach, we generate flows of packets instead of simulating packet events. We model two different types of traffic: Background traffic and Attacker traffic. Attacker traffic is the traffic generated and controlled by the Attacker. It is deterministic in nature, i.e., the Attacker can precisely control these flows. Background traffic is the aggregate of all other traffic in the network and is stochastic in nature. For background traffic, we assume that different flows are statistically independent.

We now describe the method for determining the input and output flows at each router. The input flow at a router at any instant is the sum of deterministic traffic (from the Attacker) and the stochastic traffic (background traffic). The input deterministic traffic at a given router is the sum of outgoing deterministic flows from all routers connected to the given router. The input stochastic flows are generated using Gaussian copulas [25, 14]. Let Z be a discrete-time stochastic process describing a flow; the random variable Z_t is the flow rate at time t. Let ρ_{st} be the correlation coefficient between Z_s and Z_t ; we assume that $\rho_{st} = 0$ for $|s{-}t| > n$, where n is a positive integer. Let -

$$
\begin{aligned}
\Sigma &\equiv \{\rho_{st}\},\ s,t = i-1, i-2, \ldots, i-n,\ (n \times n \text{ matrix}),\\
\underline{\rho}_{ji} &\equiv (\rho_{i,i-1}, \rho_{i,i-2}, \ldots, \rho_{i,i-n}),\ (1 \times n \text{ vector}),\\
\underline{\rho}_{ij} &\equiv \underline{\rho}_{ji}^{T}.
\end{aligned}
$$

Let $\underline{Z}_i \equiv (Z_{j-1}, \ldots, Z_{j-n})^T$. We have that $Z_j \sim \mathbb{N}(\mu, \sigma^2)$, i.e., Z_j is a Gaussian random variable with mean μ and variance σ^2. Then we have

$$
(Z_j | \underline{Z}_i = \underline{z}_i) = \mathbb{N}(\tilde{\mu}, \tilde{\rho}\sigma^2), \tag{3.3}
$$

where

$$
\begin{aligned}
\tilde{\mu} &= \underline{\rho}_{ji} \Sigma^{-1} \underline{z}_i\\
\tilde{\rho} &= 1 - \underline{\rho}_{ji} \Sigma^{-1} \underline{\rho}_{ij}.
\end{aligned}
$$

In this simulation, we choose $n = 5$, $\rho_{st} = (0.75)^{|s-t|}$, $\mu = 0.8 \times \frac{outgoing\_bandwidth}{number\_of\_flows}$, and $\sigma^2 = \mu/3$.

Given the input flows at a router the *Drop_Rate* is defined as $Outgoing\_Bandwitdh \times (1 + Target\_Drop\_Rate) - (Input\_Flowrate)$. In absence of filtering, the number of dropped packets from any flow at time t is $Drop\_Rate \times t$. If there is a filter on any flow, then the flow rate is scaled by the filter rate before the *Drop_Rate* is calculated. In the simulation each router periodically samples its stochastic input flows and recalculates its output flows. Any changes in deterministic output flows are sent to routers

connected to the given router. A router on receiving an update in deterministic input flow, recalculates its output flows and updates connected routers. The dropped packets set is also updated at each recalculation of flows at the routers. Since the pushback mechanism only needs the dropped-packets set and the test for congestion to work, the mechanism works well with the flow-based simulation.

3.3 Simulation Results

In this section, we present the simulation results of both packet-based simulation and flow-based simulation.

3.3.1 Packet-based Simulation Results

The results from payoff measurements are presented in *Table* 3.3. We use the parameters given in Section 3.2. The payoff matrices of the System and the Attacker are shown in *Table* 3.4 and *Table* 3.5.

The Nash Equilibrium calculated using Gambit [24] is $(0, 0, 1, 0, 0)$ for the Attacker and $(0, 0, 0, 1, 0, 0)$ for the System. The fictitious play simulation results are given in *Fig.* 3.3 and *Fig.* 3.4. The number of time steps simulated is about 300. It can be seen that the frequency of pure strategy S_4 of the System goes to 1 while those of other pure strategies go to 0. Similarly, the frequency of pure strategy A_3 goes to 1, while those of the others go to 0. This coincides with the Nash Equilibrium obtained from Gambit.

Actions	Attacker Traffic	Good Traffic	Poor Traffic	Attacker Payoff	System Payoff
A1,S1	2.42E+05	5.15E+05	8.70E+03	0.6067	0.3933
A1,S2	2.42E+05	5.12E+05	8.48E+03	0.6083	0.3917
A1,S3	3.07E+05	5.00E+05	9.22E+03	0.6194	0.3806
A1,S4	2.40E+05	5.33E+05	1.55E+04	0.5961	0.4039
A1,S5	3.66E+05	5.49E+05	8.52E+03	0.6051	0.3949
A1,S6	2.40E+05	5.58E+05	7.66E+03	0.5892	0.4108
A2,S1	2.41E+05	5.21E+05	8.04E+03	0.6044	0.3956
A2,S2	2.41E+05	5.17E+05	8.27E+03	0.6058	0.3942
A2,S3	3.06E+05	5.05E+05	8.43E+03	0.6174	0.3826
A2,S4	2.41E+05	5.33E+05	1.50E+04	0.5967	0.4033
A2,S5	3.68E+05	5.54E+05	7.03E+03	0.6039	0.3961
A2,S6	2.39E+05	5.77E+05	6.70E+03	0.5817	0.4183
A3,S1	2.42E+05	5.12E+05	9.17E+03	0.6080	0.3920
A3,S2	2.43E+05	5.09E+05	9.26E+03	0.6090	0.3910
A3,S3	3.07E+05	5.06E+05	1.07E+04	0.6159	0.3841
A3,S4	2.41E+05	5.24E+05	1.65E+04	0.5995	0.4005
A3,S5	3.68E+05	5.46E+05	1.16E+04	0.6053	0.3947
A3,S6	2.45E+05	5.30E+05	7.99E+03	0.6012	0.3988
A4,S1	2.44E+05	9.78E+05	1.51E+05	0.3556	0.6444
A4,S2	2.44E+05	9.78E+05	1.51E+05	0.3556	0.6444
A4,S3	2.66E+05	9.68E+05	1.64E+05	0.3563	0.6437
A4,S4	2.35E+05	9.82E+05	1.46E+05	0.3551	0.6449
A4,S5	2.87E+05	9.56E+05	1.79E+05	0.3575	0.6425
A4,S6	2.52E+05	9.74E+05	1.56E+05	0.3560	0.6440
A5,S1	2.17E+05	8.98E+05	7.78E+05	0.1256	0.8744
A5,S2	2.17E+05	8.98E+05	7.78E+05	0.1256	0.8744
A5,S3	1.77E+05	1.04E+06	6.50E+05	0.1172	0.8828
A5,S4	2.19E+05	8.64E+05	7.80E+05	0.1393	0.8607
A5,S5	2.21E+05	8.33E+05	7.80E+05	0.1521	0.8479
A5,S6	2.15E+05	9.32E+05	7.76E+05	0.1122	0.8878

Table 3.3: Payoff Measurements - Network I

System \ Attacker	A_1	A_2	A_3	A_4	A_5
S_1	0.3933	0.3956	0.3920	0.6444	0.8744
S_2	0.3917	0.3942	0.3910	0.6444	0.8744
S_3	0.3806	0.3826	0.3841	0.6437	0.8828
S_4	0.4039	0.4033	0.4005	0.6449	0.8607
S_5	0.3949	0.3961	0.3947	0.6425	0.8479
S_6	0.4108	0.4183	0.3988	0.6440	0.8878

Table 3.4: System's Payoff Matrix - Network I

Attacker \ System	S_1	S_2	S_3	S_4	S_5	S_6
A_1	0.6067	0.6083	0.6194	0.5961	0.6051	0.5892
A_2	0.6044	0.6058	0.6174	0.5967	0.6039	0.5817
A_3	0.6080	0.6090	0.6159	0.5995	0.6053	0.6012
A_4	0.3556	0.3556	0.3563	0.3551	0.3575	0.3560
A_5	0.1256	0.1256	0.1172	0.1393	0.1521	0.1122

Table 3.5: Attacker's Payoff Matrix - Network I

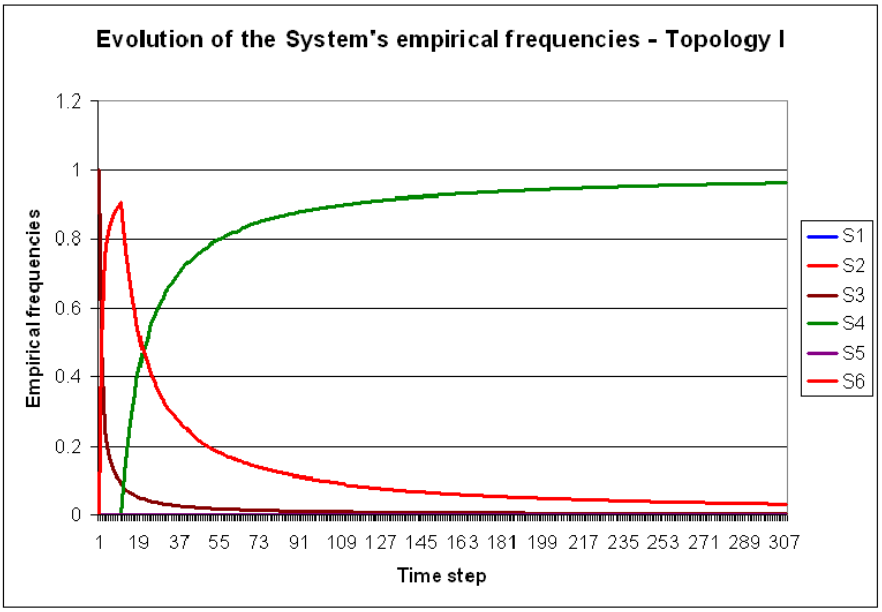

Figure 3.3: System's Empirical Frequencies - Network I

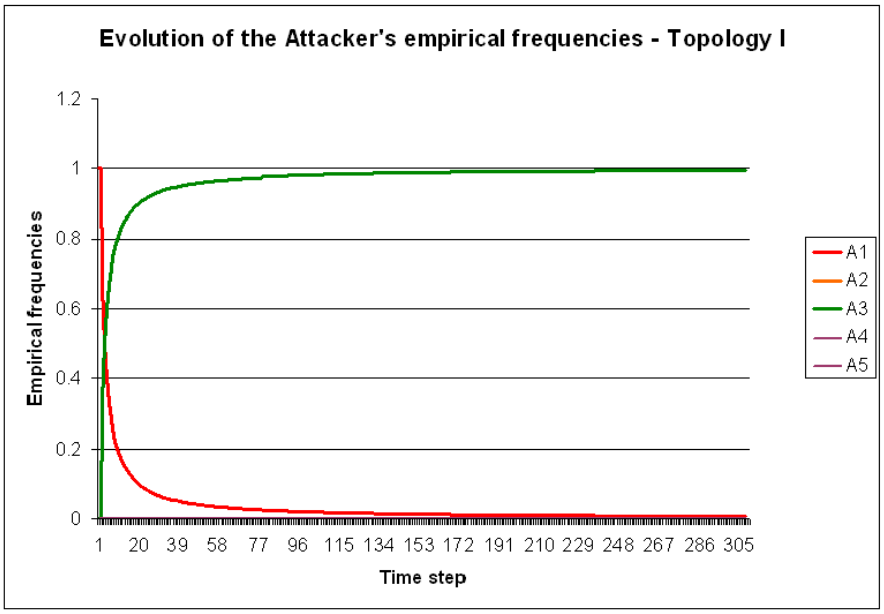

Figure 3.4: Attacker's Empirical Frequencies - Network I

3.3.2 Flow-based Simulation Results

The results from payoff measurements and the payoff matrices of the System and the Attacker are presented in *Table* 3.6, *Table* 3.7, and *Table* 3.8, respectively. Again, we use the parameters given in *Section* 3.2.

From Gambit, there are three Nash equilibria For Network II:

- Attacker $(0, 0, 0, 1, 0)$, System $(0, 0.992, 0, 0, 0.008, 0)$.
- Attacker $(0, 0, 0, 1, 0)$, System $(0, 0.992, 0.008, 0, 0, 0)$.
- Attacker $(0, 0, 0, 1, 0)$, System $(0, 1, 0, 0, 0, 0)$.

Again, the fictitious play simulation results are in agreement with the Nash equilibria obtained from Gambit as can be seen in Fig. 3.5 and Fig. 3.6. Note

Actions	Attacker	Background Traffic	Attacker Payoff	System Payoff
A1,S1	1.25E+08	3.03E+07	0.9665	0.2335
A1,S2	1.25E+08	3.03E+07	0.9665	0.2335
A1,S3	1.27E+08	2.86E+07	0.9796	0.2204
A1,S4	1.25E+08	3.03E+07	0.9659	0.2341
A1,S5	1.27E+08	2.86E+07	0.9796	0.2204
A1,S6	1.25E+08	3.01E+07	0.9676	0.2324
A2,S1	1.25E+08	3.03E+07	0.9665	0.2335
A2,S2	1.25E+08	3.03E+07	0.9664	0.2336
A2,S3	1.27E+08	2.86E+07	0.9796	0.2204
A2,S4	1.25E+08	3.03E+07	0.9659	0.2341
A2,S5	1.27E+08	2.86E+07	0.9796	0.2204
A2,S6	1.25E+08	3.01E+07	0.9676	0.2324
A3,S1	1.25E+08	3.03E+07	0.9664	0.2336
A3,S2	1.25E+08	3.03E+07	0.9664	0.2336
A3,S3	1.27E+08	2.86E+07	0.9796	0.2204
A3,S4	1.25E+08	3.03E+07	0.9659	0.2341
A3,S5	1.27E+08	2.86E+07	0.9796	0.2204
A3,S6	1.25E+08	3.01E+07	0.9676	0.2324
A4,S1	1.26E+08	2.94E+07	0.9729	0.2271
A4,S2	1.25E+08	3.02E+07	0.9666	0.2334
A4,S3	1.25E+08	3.02E+07	0.9666	0.2334
A4,S4	1.26E+08	2.94E+07	0.9729	0.2271
A4,S5	1.25E+08	3.02E+07	0.9666	0.2334
A4,S6	1.25E+08	3.02E+07	0.9669	0.2331
A5,S1	3.06E+07	3.46E+07	0.8169	0.3831
A5,S2	3.06E+07	3.46E+07	0.8169	0.3831
A5,S3	3.06E+07	3.46E+07	0.8169	0.3831
A5,S4	3.06E+07	3.46E+07	0.8169	0.3831
A5,S5	3.06E+07	3.46E+07	0.8169	0.3831
A5,S6	3.06E+07	3.46E+07	0.8169	0.3831

Table 3.6: Payoff Measurements - Network II

System \ Attacker	A_1	A_2	A_3	A_4	A_5
S_1	0.2335	0.2335	0.2336	0.2271	0.3831
S_2	0.2335	0.2336	0.2336	0.2334	0.3831
S_3	0.2204	0.2204	0.2204	0.2334	0.3831
S_4	0.2341	0.2341	0.2341	0.2271	0.3831
S_5	0.2204	0.2204	0.2204	0.2334	0.3831
S_6	0.2324	0.2324	0.2324	0.2331	0.3831

Table 3.7: System's Payoff Matrix - Network II

Attacker \ System	S_1	S_2	S_3	S_4	S_5	S_6
A_1	0.9665	0.9665	0.9796	0.9659	0.9796	0.9676
A_2	0.9665	0.9664	0.9796	0.9659	0.9796	0.9676
A_3	0.9664	0.9664	0.9796	0.9659	0.9796	0.9676
A_4	0.9729	0.9666	0.9666	0.9729	0.9666	0.9669
A_5	0.8169	0.8169	0.8169	0.8169	0.8169	0.8169

Table 3.8: Attacker's Payoff Matrix - Network II

that the first two mixed-strategy Nash equilibria (MSNE) shown above are very close to pure strategies. In the case where there are mixed-strategy Nash equilibria, if the first action (at time $\tau = 0$) of each player is chosen appropriately (which is necessary only if there are both mixed-strategy NE and pure-strategy NE), the empirical frequencies of the player's actions will converge to a mixed-strategy Nash equilibrium, which means each player will alternate among the pure strategies constituting the MSNE with proportional numbers of times.

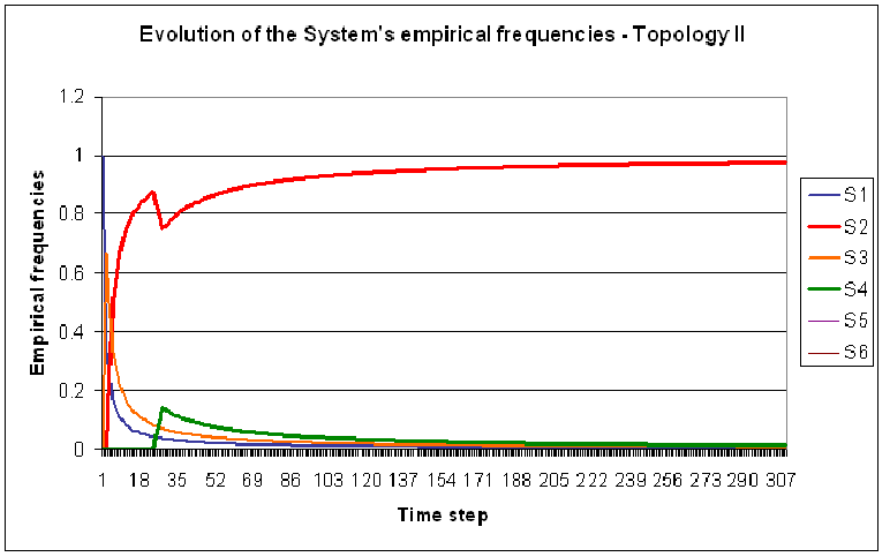

Figure 3.5: System's Empirical Frequencies - Network II

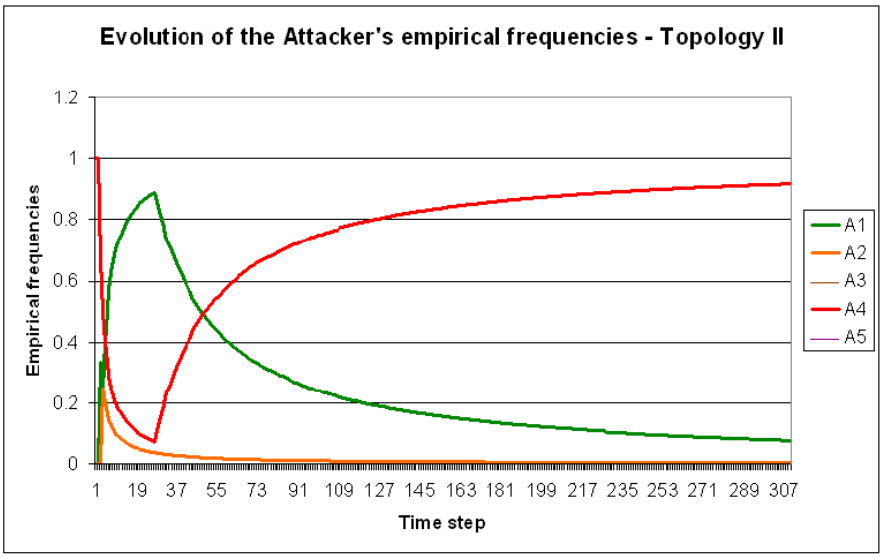

Figure 3.6: Attacker's Empirical Frequencies - Network II

Chapter 4

Defense Against Multiple Attacks

4.1 Introduction

The system developed in Chapter 3 was geared towards defense against DDoS attacks. However, a sophisticated attacker may have at his disposal multiple forms of attack. For example, even for carrying out a Denial of Service attack, an attacker has a choice between SYN-Flood, Smurf or Ping-of-Death attacks to name a few. In this chapter, we demonstrate a game-theoretic defense system against such an attacker.

The attacker under consideration is assumed to be able to carry out four different types of attacks namely SYN-Flood, Smurf, Dictionary and Portscan attack. A SYN-Flood is a form of denial-of-service attack in which an attacker sends a succession of SYN requests to a target's system. These SYN requests

lead to half-open connections on the server, thereby using up resources of the server. A Smurf attack is a type of denial-of-service attack that floods a target system via spoofed broadcast ping messages. A Dictionary attack is a technique for defeating a cipher or authentication mechanism by trying to determine its decryption key or passphrase by searching likely possibilities. A Portscan attack sends client requests to a range of server port addresses on a host, with the goal of finding an active port and exploiting a known vulnerability of that service. A detailed description of these attacks and their implementation is given in Section 4.3. Each option available to the attacker has a specific cost and benefit.

The system has specific defenses geared towards defending against each kind of attack. Like the attacker, the system too has a specific cost and benefit from each defense. Game theory is used by the system to decide the exact defense to employ at a given time. The defense mechanisms available to the system are described in Section 4.4.

Some of the attacks available to the attacker in our consideration have been around quite a while and hence are not very effective in today's networks. However, we chose these attacks to demonstrate the working of our system. We can very easily substitute these with more recent attacks (and their corresponding defenses) without changing the overall structure of the defense system.

Note that the actions of the attacker described above are various attacks, in contrast to the game used in Chapter 3 in which the actions of the attacker were 'intensities' of a single attack. Since we use a repeated game model, this

implies that the attack carried out by the attacker may differ from step to step. In Section 4.5.4 we provide a justification of why such an approach will be beneficial to an attacker. We also consider a case in which an attacker does not a play a game i.e. he uniformly carries out a single attack. We prove in Section 4.5.4 that even in such a case, the defense system can still derive an optimal payoff by unilaterally using fictitious play. The same argument holds for the system not playing a game. Thus, in our approach, the strategy that should be employed a player does not depend upon whether the opponent actually is playing a game or not.

4.2 Emulation

In contrast to the DDoS attacks used in Chapter 3 which work at the internet and transport layers, the attacks used in this chapter such as the Dictionary attacks are carried out at the application layer. Hence, instead of simulating the network as in Chapter 3, we need to emulate hosts, routers and the interconnecting network. Emulating hosts also allows us to use defense techniques which work at the application level. For the work described in this chapter, we use Virtual Network User Mode Linux (VNUML) [7, 11].

VNUML is a general-purpose, open-source, scenario-based management tool designed to help build virtual network testbeds automatically. It enables the user to define, run, and interact with scenarios made of GNU/Linux virtual machines interconnected through virtual networks running within a single physical host.

We used the network shown in Fig. 4.1 for our emulation. Each box in the figure represents a VNUML virtual machine container running Linux kernel 2.6.28.10. From now onwards, we use the term "System" to refer to the set of hosts and routers involved in defense against an attacker and the term "Attacker" to represent the set of hosts and routers carrying out an attack.

As can be seen from the figure, the System consists of a single host. The services running on the System host are **ssh, ftpd** and **httpd**. The ftpd service running on the system utilizes ftpd bundled with the Linux kernel. The httpd service uses turbo-http [16] which is a small and fast http server.

The Attacker consists of a host and a router which performs Network Address Translation (NAT). Typically, an attacker will use more than one host to carry out an attack. Since using a large number of hosts for the attacker was infeasible due to the memory restrictions, we simulated multiple attacking hosts by using ip-spoofing and NAT. Ip-spoofing was used in scenarios where a complete TCP connection was not needed. We used Scapy [6] to implement ip-spoofing. In scenarios where a complete TCP connection was needed, NAT was used to map a range of ports to eight ip addresses. As a result, even though just one virtual machine container was used for the Attacker, each attack could be carried out using eight ip addresses.

4.3 Attack Implementation

In this section, we describe the attacks available to the attacker in our work and their implementation.

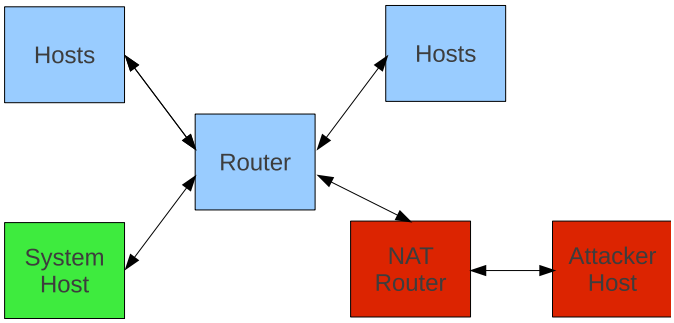

Figure 4.1: VNUML Network Topology

4.3.1 SYN-Flood

A SYN flood is a form of denial-of-service attack in which an attacker sends a succession of SYN requests to a target's system. SYN-Flood attacks are described in Section 2.1.1.1. Modern operating systems such as Linux can avoid the SYN-Flood problem by using SYN-Cookies. However, there are drawbacks of using this approach. The server must reject all TCP options (such as large windows), because the server discards the SYN queue entry where that information would otherwise be stored. A connection may also freeze when the final ACK of the three-way handshake is lost and the client first awaits data from the server (i.e. client has completed the three-way handshake, server did not receive the client's ACK and thus has not actually

opened the connection). As a result, this option is not enabled by default in the Linux kernel.

We implemented the SYN-Flood attack in Python using the Scapy [6] library. Since a full TCP connection is not required to carry out the attack, the source ip address of the SYN packets were spoofed. Modern networks typically implement egress filtering whereby ip packets with source ip-address not contained in the network are dropped. Hence, we limited the spoofed addresses to the ip addresses within the network available to the Attacker. We also carry out the attack only against http (80) port.

4.3.2 Smurf

The Smurf attack is a denial-of-service attack that generates significant computer network traffic on a victim network, thereby using up the network bandwidth. A description of Smurf attacks is given in Section 2.1.1.2.

Modern-day networks can prevent Smurf attacks through the following two steps -

1. Configuring hosts and routers not to respond to ping requests.
2. Configuring routers not to forward packets directed to broadcast addresses.

The above two steps prevent the multiplying effect on echo requests achieved by the attacker through using broadcast addresses. However, if the attacker has a sufficient number of hosts under its control, he can still generate a sizable

amount of ICMP echo-reply traffic without the need for the multiplying effect. Although such an attack is less effective, it can still cause a nuisance.

We implemented the Smurf attack in Python using Scapy [6]. To circumvent the problem of routers configured against this attack, we synthetically generated ICMP echo-reply packets and directed them towards the victim, without using a broadcast address.

4.3.3 Dictionary

A dictionary attack used to guess the password for protected resource in order to gain unauthorized access to a resource. Dictionary attacks successively try all the words in an exhaustive list called a dictionary which is a pre-arranged list of values. A description is given in Section 2.1.2.1.

In our work, the Attacker carries out a dictionary attack to guess the root password for the ftpd service run on the System.

The dictionary attack was implemented in Python using the ftp library. The dictionary for the attack consisted of all four-letter words in the English language (including the digits 1-9).

4.3.4 Portscan

As the name suggests, a Portscan attack involves 'scanning' a range of server port addresses on a host to find an active port. Once such a port has been identified, a vulnerability of the service on that port may be exploited. In our work, we use SYN scanning. A description of SYN-scanning is given in Section 2.1.3.1.

The portscan attack in our work was implemented using **nmap** which is a free and open source utility for port-scanning.

4.4 Defense Implementation

In this section we describe the actions available to the System in our work and their implementation.

4.4.1 SYN-Flood Defense

We implement a defense against a SYN-Flood attack using an iptables script. The script is shown in Algorithm 4.1. In the script, we keep a count of new connection requests (SYN packets received) from each source ip address. If the rate of requests from any ip address reaches the value of 10 per second, we immediately blacklist that ip address for 3 minutes. The result of blacklisting is that all connection requests from that ip address are dropped.

4.4.2 Smurf Defense

We implement a defense against a Smurf attack using an iptables script. The script is shown in Algorithm 4.2. In the script, we keep a count of number of ICMP echo-replies received. If the rate of echo-replies received from any ip address reaches the value of 10 per second, we immediately blacklist that ip address for 3 minutes. The result of blacklisting is that all ICMP traffic from that ip address is dropped.

Algorithm 4.1 Iptables Script for SYN-Flood Defense

```
# Create synflood chain
iptables -N synflood

# Jump to synflood chain for new connection
iptables -A FORWARD -m state --state NEW -p tcp -m tcp --syn
-j synflood

iptables -A INPUT -m state --state NEW -p tcp -m tcp --syn
-j synflood

# If hitcount is 10, jump to blacklist-addition
iptables -A synflood -m recent --name synflood-suspects
--rcheck --seconds 1 --hitcount 10 -j blacklist-180-add

# Set counter for suspects
iptables -A synflood -m recent --name synflood-suspects
--set -j RETURN
```

4.4.3 Dictionary Defense

We implement a defense against a Dictionary attack using the authentication log provided by Linux. Every authentication failure is logged by Linux in a file auth.log. We monitor this file to keep track of number of authentication failures from each ip address. If there are more than 5 authentication failures within 2 minutes from an ip address, then that ip address is blacklisted for 3 minutes. The result of blacklisting is that authentication attempts from that ip address are not accepted.

4.4.4 Portscan Defense

To detect a portscan attack, we employ two mechanisms.

Algorithm 4.2 Iptables Script for Smurf Defense

```
# Create a chain for filtering icmp echo-replies
iptables -N icmp-traffic

iptables -A INPUT -p icmp --icmp-type echo-reply
-j icmp-traffic

iptables -A FORWARD -p icmp --icmp-type echo-reply
-j icmp-traffic

# Create icmpflood chain
iptables -N icmpflood

# Add icmpflood chain to icmp-traffic
iptables -A icmp-traffic -j icmpflood

# If hitcount is 10, jump to blacklist-addition
iptables -A icmpflood -m recent --name icmpflood-suspects
--rcheck --seconds 1 --hitcount 10 -j blacklist-180-add

# Set counter for suspects
iptables -A icmpflood -m recent --name icmpflood-suspects
--set -j RETURN
```

First, we make use of Portscan Attack Detector (PSAD) [34]. PSAD is a collection of three lightweight system daemons written in Perl and C that are designed to work with the Linux Netfilter firewalling code to detect port scans and other suspect traffic. In addition, PSAD incorporates many of the TCP, UDP, and ICMP signatures included in Snort to detect highly suspect scans for various backdoor programs (e.g. EvilFTP, GirlFriend, SubSeven), DDoS tools (mstream, shaft), and advanced port scans (SYN, FIN, XMAS) which are easily leveraged against a machine via nmap. PSAD requires the iptables/netfilter to log all ip traffic.

We also use a second much simpler mechanism to detect portscan attacks. Since the Attacker uses nmap SYN scanning on all ports to carry out an attack, we monitor traffic on an arbitrary port (e.g. port 531). A client typically won't send a SYN packet to this port since no service on the system uses this port. Hence, if SYN packets are being sent to this port, it means that an Attacker is scanning all ports. This mechanism can thus be used to detect portscans.

Once a portscan has been detected, using iptables we drop all TCP traffic from the ip address from which the portscan originated.

4.5 Game Formulation

We model the interaction between the Attacker and the System as a two-player nonzero-sum repeated game. The System and the Attacker know their own payoff function but not of the adversary and engage in a Fictitious Play (FP) process as described in Section 2.2.

Action	Description
A_1	Portscan Attack
A_2	SYN-Flood Attack
A_3	Dictionary Attack
A_4	Smurf Attack

Table 4.1: Attacker's Actions

Action	Description
S_1	Portscan Defense
S_2	SYN-Flood Defense
S_3	Dictionary Defense
S_4	Smurf Defense

Table 4.2: System's Actions

4.5.1 Action Space

The Attacker has four available actions as shown in Table 4.1. The implementation of these actions is described in Section 4.3. The System also has four possible actions as shown in Table 4.2. The implementation of these actions is described in Section 4.4.

4.5.2 Payoffs

For the Attacker, the payoff from carrying out a certain attack is the algebraic sum of cost of carrying out an attack and the benefit from a successful attack.

$$
\begin{aligned}
U_{att} &= (B_{att} * I) - C_{att} \qquad (4.1)\\
where & \\
B_{att} &= Benefit\ from\ attack \\
C_{att} &= Cost\ of\ carrying\ out\ an\ attack \\
I &= \begin{cases} 1 & if\ attack\ is\ successful \\ 0 & if\ attack\ is\ unsuccessful \end{cases}
\end{aligned}
$$

For the System, the payoff as a result of choosing a certain defense is the algebraic sum of cost of the defense and the damage caused from the attack.

$$
\begin{aligned}
U_{sys} &= -C_{def} - (D_{att} * I) \qquad (4.2)\\
where & \\
C_{def} &= Cost\ of\ defense \\
D_{att} &= Damage\ caused\ by\ attack \\
I &= \begin{cases} 1 & if\ attack\ is\ successful \\ 0 & if\ attack\ is\ unsuccessful \end{cases}
\end{aligned}
$$

We used the values shown in Table 4.3 and Table 4.4 for B_{att}, C_{att}, D_{att} and C_{def} . These values were assigned by taking into consideration the effectiveness of and the costs for implementing the attacks and defenses. The payoffs for the System and the Attacker are shown in Table 4.5 and Table 4.6 respectively. It

Parameter	B_{att}	C_{att}	D_{att}
Portscan	0.35	0.15	0.15
SYN-Flood	0.5	1.0	1.0
Dictionary	0.4	1.5	1.5
Smurf	0.3	0.5	0.5

Table 4.3: Parameter Values - I

Parameter	C_{def}
Portscan Defense	0.25
SYN-Flood Defense	0.5
Dictionary Defense	0.5
Smurf Defense	0.3

Table 4.4: Parameter Values

is important to note here that the payoff functions can be changed depending upon the preferences of the Attacker and the System. If payoff functions are changed, only the NE solution to the game changes.

4.5.3 Start State

From the payoff functions for the System and the Attacker, we can see that there is no dominating strategy for either player. Hence, the first action taken by either player (before any estimate can be made of opponent's empirical frequencies) is randomly chosen from all available actions.

System/Attacker	A_1	A_2	A_3	A_4
S_1	-0.25	-1.25	-1.75	-0.75
S_2	-0.65	-0.5	-2.0	-1.0
S_3	-0.65	-1.5	-0.5	-1.0
S_4	-0.45	-1.3	-1.8	-0.3

Table 4.5: System Payoff

System/Attacker	A_1	A_2	A_3	A_4
S_1	-0.15	0.5	1.1	0.2
S_2	0.2	-0.5	1.1	0.2
S_3	0.2	0.5	-0.4	0.2
S_4	0.2	0.5	1.1	-0.3

Table 4.6: Attacker Payoff

4.5.4 Optimal Strategies

Suppose that as a result of a repeated game process of n steps, the Attacker plays action a_t^{att} at time t and the System plays action a_t^{sys} at time t. We define the payoff for a player k, $k \in \{sys, att\}$ as

$$Q_k = \frac{\sum_{t=1}^{n} U(a_t^k, a_t^{-k})}{n} \tag{4.3}$$

In other words, the payoff for a player is defined to be the average of payoffs obtained at each step of the repeated game.

By definition, a mixed-strategy NE specifies a mixed strategy for each player, in such a way that each player's mixed strategy yields the player at least as high an expected payoff as any other mixed strategy of the player, given the mixed strategies of the other players. Hence, by playing a mixed strategy given by NE, the Attacker can expect atleast as high a payoff as obtained by playing any other mixed strategy, given the strategy of the System. Therefore, the Attacker should ideally play a mixed strategy given by NE. As a consequence, provided that empirical frequencies in a FP process converge to a NE, the Attacker should employ FP. The same arguments hold for the System employing FP.

It is possible that an attacker may not actually engage in playing a game and may simply carry out one attack uniformly. This is equivalent to the Attacker choosing to play the same action at each step of a repeated game. Even in such a case, the System should employ the FP approach to maximize its payoff. The reason for this is as follows -

Without loss of generality, suppose the Attacker uniformly carries out a Portscan attack i.e. it plays action A_1 at each step of the game. Now suppose the System follows the FP approach. At time $t = 0$, System will play a random action. At time $t = 1$, empirical frequency observed by the System will be $(1, 0, 0, 0)$. It will respond with the best response to the observed empirical frequency. For the payoffs given in Table 4.5, the best response is System playing action S_1. At time $t = 1$, Attacker will again play A_1. At $t = 2$, the empirical frequency (without normalization) observed by the System will be $(2, 0, 0, 0)$. The best response to this is again System playing action S_1. At time $t = 2$, Attacker will once again play A_1. At $t = 3$, empirical frequency is $(3, 0, 0, 0)$ and best response is S_1. This process will continue upto infinity. Ignoring the initial random action, the payoff to the System is $(-0.25 * n)/n = -0.25$. From Table 4.5, we can see that this is the best payoff possible for the System when Attacker is playing action A_1 and hence, the strategy adopted by the System is optimal. A similar argument can be made for the Attacker playing other actions uniformly.

The same logic also applies to the Attacker when the System is not playing a game. Hence, a FP approach is beneficial to a player irrespective of whether the other player is actually playing the game.

4.6 Fictitious Play Simulation

The game-play proceeds via a Fictitious Play (FP) process as described in Section 2.2. The FP process is simulated in VNUML. As a part of FP simulation, when a player selects a particular action to take, we actually carry out that action in VNUML. For example, if the Attacker selects SYN-Flood as its action, the Attacker host in VNUML carries out a SYN-Flood attack. Similarly, if the System selects Portscan defense as its action, the System host in VNUML deploys a Portscan defense. Thus, we demonstrate the working of the game-theoretic defense-system through the FP simulation in VNUML.

The time step for each action is 180 seconds. We run FP simulation for two scenarios. In the first scenario, both the System and the Attacker employ FP and the action chosen at each step is the best response to the observed empirical frequencies of the opponent. In the second scenario, the System employs FP but the Attacker uniformly carries out a SYN-Flood attack. The observations of opponent's actions and calculation of empirical frequencies is described in Section 4.6.1 and the results of the simulations are described in Section 4.6.2.

4.6.1 Empirical Frequency Measurement

In order for Fictitious Play approach to work, each player must be able to observe the opponent's action. For the System, observing the action taken by the Attacker is straight-forward because the location of the attack is the system. The System can detects the action taken by the Attacker by analyzing

the authentication and iptables/netfilter logs.

Detecting the action taken by the System is not easy for the Attacker. This is because the Attacker does not (normally) have access to logs on the system where the defense action is implemented. Hence, the Attacker has to infer the action taken by the System based on the "side-effects" of the action.

In our work, the Attacker detects the action taken by the System by using a "decoy" attack. The decoy attack is a low-intensity attack such that the cost and the effectiveness of the attack is almost negligible but the intensity is enough to produce a side-effect of the action taken by the System. Our decoy attack procedure works as follows -

1. Execute all four actions of the attacker namely Portscan, SYN-Flood (on port 80), Dictionary and Smurf with a very low intensity.

2. If the action taken by System is -

   (a) Portscan Defense - All ports on the System will be filtered.

   (b) SYN-Flood - Port 80 (http) will be filtered.

   (c) Dictionary - Port 21 (ftp) will be filtered.

   (d) Smurf - No port will be filtered.

3. Scan port 21, 80 and some randomly chosen port to detect which of (a), (b), (c) or (d) is applicable.

4. Infer the action of the System using result of step 3.

This procedure works because the side-effects of the System's actions are mutually exclusive. However, if this was not the case, some other technique will have to be used to detect the action of the System.

4.6.2 Results

The NE solution to the single-shot game, calculated using Gambit [24], is

- System - (0.044, 0.315, 0.610, 0.029)
- Attacker - (0.311, 0.297, 0.198, 0.194)

The payoff when both players use NE strategies is

- System : -0.940
- Attacker: 0.185

The empirical frequencies resulting from fictitious play simulation for the first scenario (both players employ FP) are shown in Fig. 4.2 and Fig. 4.3. The game was simulated for 10^5 time steps. From the figures, we can see that empirical frequencies converge to NE equilibrium strategies. The resultant payoff (as defined in Eq. 4.3) is -0.986 for the System and 0.184 for the Attacker which is very close to the NE payoff. The observed deviation from NE payoff can be attributed to the time taken in FP for convergence of empirical frequencies to NE.

In the second scenario, the Attacker uniformly carries out a SYN-Flood attack (i.e. plays action A_2) while the System employs FP. The resultant empirical frequencies for the System are shown in Fig. 4.4. From the figure,

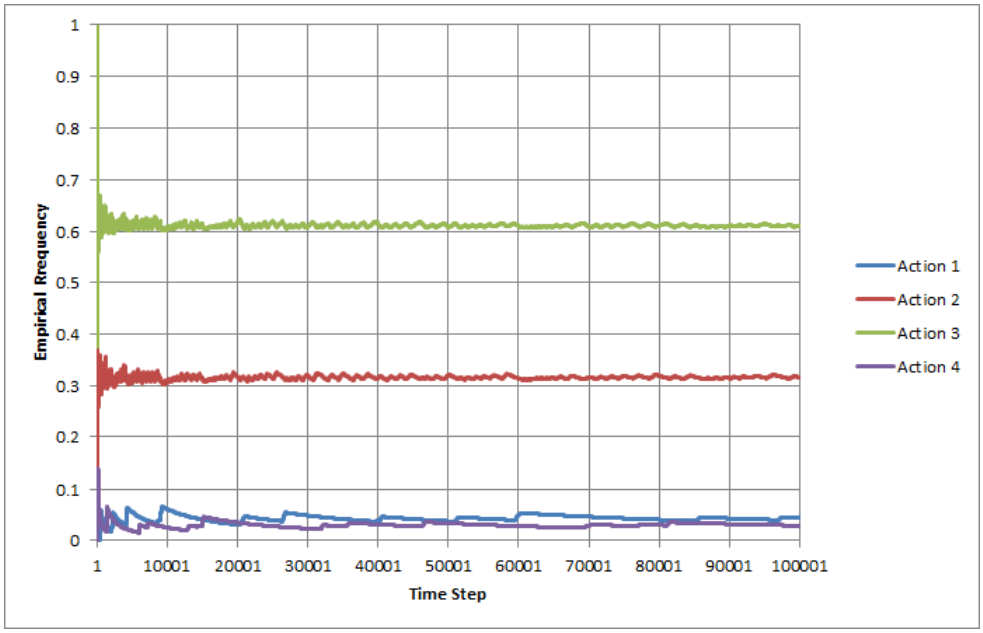

Figure 4.2: System's Empirical Frequencies in Scenario-I

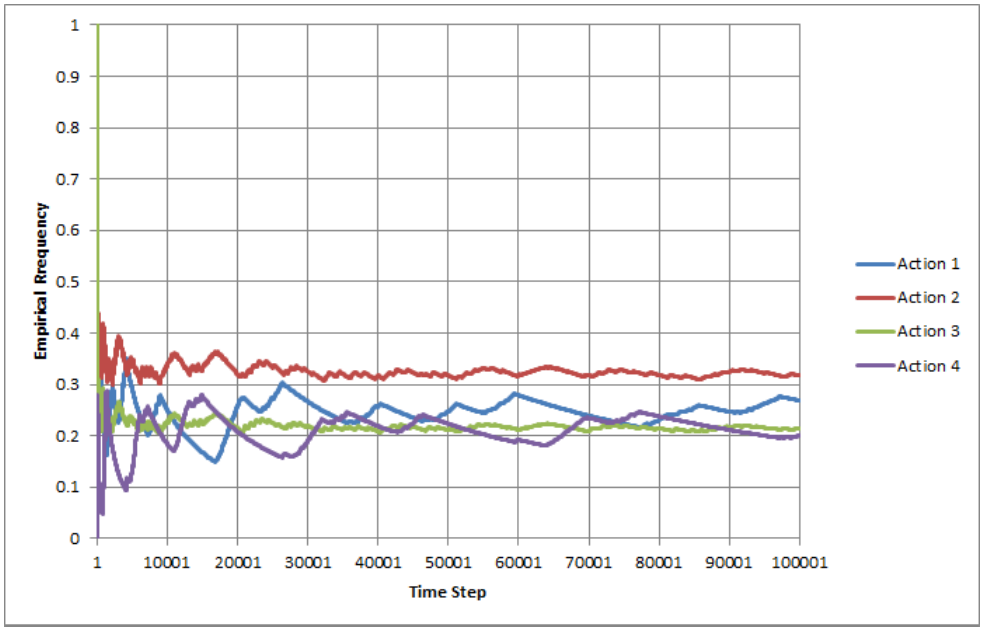

Figure 4.3: Attacker's Empirical Frequencies in Scenario - I

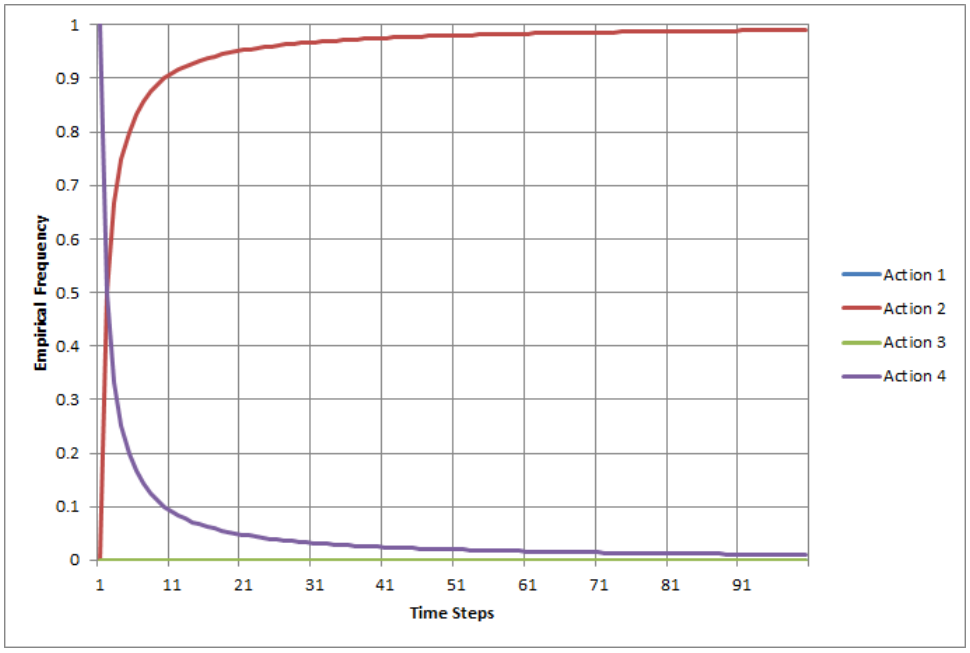

Figure 4.4: System's Empirical Frequencies in Scenario - II

we can see that empirical frequency for action S_2 converges to 1 while empirical frequencies for other actions converge to 0. The resultant payoff for the System is (ignoring payoff for the first random action) is -0.5, which is the maximum possible payoff for the System if Attacker plays action A_2. The payoff is also much better than NE-payoff of -0.940. The resultant payoff for the Attacker is -0.5 which is much worse than NE payoff of 0.184. This shows that it is always beneficial for a player to engage in FP.

Chapter 5

Conclusion

In this thesis, we have demonstrated the use of game-theory in devising effective defense mechanisms against network attacks. We have developed game-theoretic defenses addressing two scenarios namely DDoS attacks and a combination of SYN-Flood, Smurf, Portscan and Dictionary attacks. We have modeled attacks and defense-mechanisms as a repeated two-player nonzero sum game between an attacker and a defense system.

To address the fact that an attacker and a defense-system is not likely to know objectives and payoff functions of adversaries, we have used a fictitious play mechanism (classical FP) that allows players to learn their Nash Equilibrium strategies without necessarily knowing the opponent's payoff function. While using FP does not guarantee convergence to Nash Equilibrium, for several classes of games, it has been shown that such a FP process will finally render both players playing their Nash equilibrium strategies. For the game formulations that we have used, FP process does converge to NE.

For the scenario in which an attacker carries out a DDoS attack, we modeled the actions of the attacker as intensities or datarates employed in carrying out the attack. The actions of the defense-system consist of values of parameters involved in a *Pushback* defense mechanism. Using PRIME SSF, we simulated FP in this scenario. For the simulation we used two network topologies. The first network used Megabit speed links while the second network used Gigabit speed links. Since simulating each packet-event in a Gigabit speed network was computationally very expensive in PRIME SSF, we used a flow-based simulation for the Gigabit speed network. In the flow-based simulation, we simulated flow events instead of packet events. For both simulations, FP resulted in empirical frequencies converging to NE strategies.

For the scenario in which an attacker possesses ability to carry out a number of different types of attacks, we modeled the actions of the attacker as the attack options available to him. The actions of the defense-system consisted of defense-mechanisms available to the system where each mechanism was geared towards mitigating a particular type of attack. We used VNUML to emulate hosts and routers running a Linux kernel. The emulated machines were then used to carry out attacks and deploy defense mechanisms against attacks. We tested the working of the game-theoretic defense system within VNUML for two cases - one in which the attacker plays a game and the other in which the attacker does not play a game but carries out a single attack uniformly. For both cases, we were able to demonstrate that it is beneficial for the the defense system to employ FP.

Through the two scenarios, we have successfully demonstrated effectiveness

of a game-theoretic defense system. Though some of the attacks and defense mechanisms that we have employed are not very effective by themselves, we can easily substitute them with more effective mechanisms, without changing the manner in which game-theory is used to combine these mechanisms.

Bibliography

[1] The netfilter project. `http://www.netfilter.org`.

[2] Samuel Abdelsayed, David Glimsholt, Christopher Leckie, Simon Ryan, and Samer Shami. An efficient filter for denial-of-service bandwidth attacks. In *In IEEE Global Telecommunications Conference(GLOBECOM 2003)*, pages 1353–1357, 2003.

[3] Tansu Alpcan and Tamer Basar. An intrusion detection game with limited observations. In *12th International Symposium on Dynamic Games and Applications*, Sophia Antipolis, France, July 2006.

[4] Ulrich Berger. Fictitious play in 2xn games. Game theory and information, EconWPA, March 2003.

[5] D.J. Bernstein. Syn cookies. `http://cr.yp.to/syncookies.html`, 1996. [Online; accessed 5-April-2011].

[6] Philippe Biondi. Scapy - packet manipulation tool. `http://www.secdev.org/projects/scapy/`, 2011. [Online; accessed 5-April-2011].

[7] Departamento de Ingenieria de Sistemas Telematicos (DIT). Virtual network user mode linux. `http://www.dit.upm.es/vnuml`, 2011. [Online; accessed 5-April-2011].

[8] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[9] Wesley M. Eddy. Defenses against tcp syn flooding attacks. *Internet Protocol Journal*, 9(4), 2006.

[10] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. `http://www.ietf.org/rfc/rfc2827.txt`, 2000.

[11] Fermin Galan, David Fernandez, Walter Fuertes, Miguel Gomez, and Jorge Lopez de Vergara. Scenario-based virtual network infrastructure management in research and educational testbeds with vnuml. *Annals of Telecommunications*, 64:305–323, 2009. 10.1007/s12243-009-0104-3.

[12] Thomer M. Gil and Massimiliano Poletto. Multops: a data-structure for bandwidth attack detection. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, pages 3–3, Berkeley, CA, USA, 2001. USENIX Association.

[13] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *In Proceedings of Network and Distributed System Security Symposium*, 2002.

[14] Dong Jin. Qualifying examination written report – an efficient gigabit ethernet switch model for large-scale simulation. Technical report, University of Illinois at Urbana-Champaign, 2010.

[15] Tanmay Khirwadkar, Kien C. Nguyen, David M. Nicol, and Tamer Basar. Methodologies for evaluating game theoretic defense against ddos attacks. In *Winter Simulation Conference*, pages 697–707, 2010.

[16] ACME Labs. thttpd - http server. `http://www.acme.com/software/thttpd/`, 2011. [Online; accessed 5-April-2011].

[17] Jun Li, J. Mirkovic, Mengqiu Wang, P. Reiher, and Lixia Zhang. Save: source address validity enforcement protocol. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1557 – 1566, june 2002.

[18] Michael Liljenstam, Jason Liu, David Nicol, Yougu Yuan, Guanhua Yan, and Chris Grier. Rinse: the real-time immersive network simulation environment for network security exercises. In *In Proceedings of the 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS*, 2005.

[19] Jason Liu. Parallel real-time immersive modeling environment (prime) scalable simulation framework (ssf). Technical report, Colorado School of Mines, 2006.

[20] Jason Liu. The design of prime ssfnet: A tutorial for developers of the network simulator. Technical report, Colorado School of Mines, 2007.

[21] Peng Liu and Wanyu Zang. Incentive-based modeling and inference of attacker intent, objectives, and strategies. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 179–189, New York, NY, USA, 2003. ACM.

[22] Yu Liu and Hong Man. A bayesian game approach for intrusion detection in wireless ad hoc networks. *Workshop on Game Theory for Communications and Networks*, 2006.

[23] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32:62–73, July 2002.

[24] Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory. Technical report, Version 0.2007.01.30, 2006.

[25] R. B. Nelsen. *An Introduction to Copulas*. Springer Series in Statistics. Springer-Verlag, New York, NY, 2006. 2nd Edition.

[26] K. C. Nguyen, T. Alpcan, and T. Başar. Fictitious play with time-invariant frequency update for network security. In *Proc. of the 2010 IEEE Multi-Conference on Systems and Control (MSC10)*, Yokohama, Japan, September 2010.

[27] K.C. Nguyen, T. Alpcan, and T. Basar. Security games with incomplete information. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1 –6, 2009.

[28] K.C. Nguyen, T. Alpcan, and T. Basar. Security games with decision and observation errors. In *American Control Conference (ACC), 2010*, 30 2010.

[29] David M. Nicol and Guanhua Yan. High-performance simulation of low-resolution network flows. *Simulation*, 82(1):21–42, 2006.

[30] Kihong Park and Heejo Lee. On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. *SIGCOMM Comput. Commun. Rev.*, 31:15–26, August 2001.

[31] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Defending against distributed denial of service attacks using selective pushback. In *In Proceedings of the Ninth IEEE International Conference on Telecommunications (ICT*, pages 411–429, 2002.

[32] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Protection from distributed denial of service attack using history-based ip filtering. pages 482–486, 2003.

[33] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv.*, 39, April 2007.

[34] Michael Rash. Psad - portscan attack detector. `http://www.cipherdyne.org/psad/`, 2011. [Online; accessed 5-April-2011].

[35] Julia Robinson. An iterative method of solving a game. *Ann. Math.*, 54:296–301, 1951.

[36] Karin Sallhammar, Bjarne E. Helvik, and Svein J. Knapskog. On stochastic modeling for integrated security and dependability evaluation. *Journal of Networks*, 1, 2006.

[37] Jeff S. Shamma and Gurdal Arslan. Unified convergence proofs of continuous-time fictitious play. *IEEE Transactions on Automatic Control*, 49(7):1137–1142, July 2004.

[38] Jeff S. Shamma and Gurdal Arslan. Dynamic fictitious play, dynamic gradient play, and distributed convergence to nash equilibria. *IEEE Transactions on Automatic Control*, 50(3):312–327, March 2005.

[39] Haining Wang, Danlu Zhang, and Kang G. Shin. Detecting syn flooding attacks. In *In Proceedings of the IEEE Infocom*, pages 1530–1539. IEEE, 2002.