2010

Application of information theory and statistical learning to anomaly detection

Steven Gianvecchio
College of William & Mary - Arts & Sciences

Recommended Citation

Gianvecchio, Steven, "Application of information theory and statistical learning to anomaly detection" (2010). *Dissertations, Theses, and Masters Projects.* Paper 1539623563.
https://dx.doi.org/doi:10.21220/s2-8c4b-sg87

Application of Information Theory and Statistical Learning to Anomaly Detection

Steven Gianvecchio

Rochester, New York

Master of Science, College of William and Mary, 2006
Bachelor of Science, State University of New York at Brockport, 2001

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
May 2010

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

[illegible]

Steven Gianvecchio

Approved by the Committee, April 2010

[illegible]

Committee Chair
Associate Professor Haining Wang, Computer Science
The College of William and Mary

[illegible]

Associate Professor Phil Kearns, Computer Science
The College of William and Mary

[illegible]

Professor Evgenia Smirni, Computer Science
The College of William and Mary

Weizhen Mao

Associate Professor Weizhen Mao, Computer Science
The College of William and Mary

[illegible]

Professor Chi-Kwong Li, Mathematics
The College of William and Mary

ABSTRACT PAGE

In today's highly networked world, computer intrusions and other attacks area constant threat. The detection of such attacks, especially attacks that are new or previously unknown, is important to secure networks and computers. A major focus of current research efforts in this area is on anomaly detection.

In this dissertation, we explore applications of information theory and statistical learning to anomaly detection. Specifically, we look at two difficult detection problems in network and system security, (1) detecting covert channels, and (2) determining if a user is a human or bot. We link both of these problems to entropy, a measure of randomness, information content, or complexity, a concept that is central to information theory. The behavior of bots is low in entropy when tasks are rigidly repeated or high in entropy when behavior is pseudo-random. In contrast, human behavior is complex and medium in entropy. Similarly, covert channels either create regularity, resulting in low entropy, or encode extra information, resulting in high entropy. Meanwhile, legitimate traffic is characterized by complex interdependencies and moderate entropy. In addition, we utilize statistical learning algorithms, Bayesian learning, neural networks, and maximum likelihood estimation, in both modeling and detecting of covert channels and bots.

Our results using entropy and statistical learning techniques are excellent. By using entropy to detect covert channels, we detected three different covert timing channels that were not detected by previous detection methods. Then, using entropy and Bayesian learning to detect chat bots, we detected 100% of chat bots with a false positive rate of only 0.05% in over 1400 hours of chat traces. Lastly, using neural networks and the idea of human observational proofs to detect game bots, we detected 99.8% of game bots with no false positives in 95 hours of traces. Our work shows that a combination of entropy measures and statistical learning algorithms is a powerful and highly effective tool for anomaly detection.

Table of Contents

To Mom, Dad, Cheryl, Michelle, and Mike.

“Little by little, one travels far.” – J.R.R. Tolkien

ACKNOWLEDGMENTS

This thesis would not have been possible without the assistance and support of many people. I would like to first and foremost extend my deepest appreciation to my research advisor, Haining Wang, for his constant help and guidance with my research. His focus on important research problems and high quality research has greatly impacted my research and career development.

I would like to thank Phil Kearns, Evgenia Smirni, Weizhen Mao, and Chi-Kwong Li for serving on my thesis committee and for their valuable comments, suggestions, and feedback. I would also like to thank the staff of the Department of Computer Science for all of their assistance. In particular, I would like to especially thank Vanessa Godwin and Jacqulyn Johnson.

I would also like to express my appreciation to various friends, fellow graduate students, and research collaborators that helped with different stages of my research, Mengjun Xie, Zhenyu Wu, Sushil Jajodia, Zi Chu, Aaron Polaski, Sathish Indika, Lee McDaniel, and others. I feel very fortunate to have worked with such brilliant people.

Last, and most important, I would like to extend my deepest gratitude to my parents, Tom and Lorraine, and to my brother and sisters, Mike, Cheryl, and Michelle, for their support and encouragement over the years while I completed my Ph.D. studies.

List of Tables

List of Figures

Application of Information Theory and Statistical Learning to Anomaly Detection

Chapter 1

Introduction

As the world continues to become increasingly connected, the number of computer intrusions and other attacks continues to grow. As the number of attacks grows, a shift in hackers' motivation from "hacking for-fun" to "hacking for-profit" has made attacks more sophisticated and more dangerous than ever. Indeed, in 2008, reports show that the volume of malware grew almost three-fold [78] and attacks against US government networks increased by 39.7% [37]. With the threat of attack rising, detecting attacks—especially attacks that are new and previously unknown—is critical in securing networks and computers.

In light of this trend, a major focus of current research efforts is on anomaly detection. The term anomaly detection refers to detecting patterns (called anomalies) that fall outside of the normal behavior in a system. The main benefit of anomaly detection is in detecting novel or so-called "zero-day" attacks, which other detection methods often fail to detect. As systems are most vulnerable to unknown attacks, anomaly detection is a critical component in securing systems against them. The main downside to anomaly detection is false alarms, i.e., anomalies that are not real threats.

In this dissertation, we explore different applications of information theory and statistical learning to anomaly detection. With statistical learning often being a form of information extraction, information theory and statistical learning are closely related. Specifically, this dissertation addresses two challenging detection problems from network and system security, (1) detecting covert channels, and (2) determining if a user is a human or a bot. While covert channels are a classic problem, dating back to at least the 1970s, bots are a very new problem, only from the last decade or so. Although two very distinct problems, both can be linked to entropy—a measure of randomness, information content, or complexity—a central concept in information theory. In two separate studies, detailed in Chapters 3 and 4, we are able to link covert channels and bots to entropy. The behavior of bots is low in entropy when their tasks are rigidly repeated or high in entropy when components of their behavior are randomized. By comparison, the behavior of humans is complex and medium in entropy. In other words, bots are simple and predictable, whereas humans are much more complicated. Likewise, covert channels either increase regularity, resulting in lower entropy, or add additional information, resulting in higher entropy. At the same time, legitimate (non-covert) traffic is characterized by complex interdependencies and moderate entropy. In short, both bots and covert channels can be described by either high or low entropy. Additionally, we utilize statistical learning, including Bayesian learning, neural networks, and maximum likelihood estimation, to address both problems.

To develop better anomaly detection techniques, especially for covert channels and bots, we investigate different ways of using entropy measures and statistical learning techniques, both separate and in combination, for modeling and detecting of covert channels and bots. For covert channels, we design a covert timing channel, and model, simulate, and test three different covert timing channels, and then, we propose an entropy-based approach for detecting covert timing channels. For bots, we start by performing a large-scale measurement study of chat bots, and then, we propose a hybrid classification system, based on entropy and statistical learning for detecting chat bots. In addition, we characterize game playing in a popular online game, and then, we propose a game bot defense system, based on statistical learning and the idea of human observational proofs for detecting online game bots. Our research contributions are summarized as follows:

1) Designing and Modeling Covert Timing Channels

The exploration of advanced covert timing channel design is important to understand and defend against covert timing channels. This chapter introduces a new class of covert timing channels, called model-based covert timing channels, which exploit the statistical properties of legitimate network traffic to evade detection in an effective manner. We design and implement an automated framework for building model-based covert timing channels. The framework consists of four main components: filter, analyzer, encoder, and transmitter. The filter characterizes the features of legitimate network traffic, and the analyzer fits the observed traffic behavior to a model. Then, the encoder and transmitter use the model to generate covert traffic and blend with legitimate network traffic. The framework is lightweight, and the overhead induced by model fitting is negligible. To validate the effectiveness of the proposed framework, we conduct a series of experiments in LAN and WAN environments. The experimental results show that model-based covert timing channels provide a significant increase in detection resistance with only a minor loss in capacity.

2) Detecting Covert Timing Channels

The detection of covert timing channels is of increasing interest in light of recent practice on the exploitation of covert timing channels over the Internet. However, due to the high variation in legitimate network traffic, detecting covert timing channels is a challenging task. The existing detection schemes are ineffective to detect most of the covert timing channels known to the security community. This chapter introduces a new entropy-based approach to detecting various covert timing channels. Our new approach is based on the observation that the creation of a covert timing channel has certain effects on the entropy of the original process, and hence, a change in the entropy of a process provides a critical clue for covert timing channel detection. Exploiting this observation, we investigate the use of entropy and conditional entropy in detecting covert timing channels. Our experimental results show that our entropy-based approach is sensitive to the current covert timing channels, and is capable of detecting them in an accurate manner.

3) Measurement and Classification of Chat Bots

The abuse of chat services by automated programs, known as chat bots, poses a serious threat to Internet users. Chat bots target popular chat networks to distribute spam and malware. In this chapter, we first conduct a series of measurements on a large commercial chat network. Our measurements capture a total of 14 different types of chat bots ranging from simple to advanced. Moreover, we observe that human behavior is more complex than bot behavior. Based on the measurement study, we propose a classification system to accurately distinguish chat bots from human users. The proposed classification system consists of two components: (1) an entropy-based classifier and (2) a machine-learning-based classifier. The two classifiers complement each other in chat bot detection. The entropy-based classifier is more accurate to detect unknown chat bots, whereas the machine-learning-based classifier is faster to detect known chat bots. Our experimental evaluation shows that the proposed classification system is highly effective in differentiating bots from humans.

4) Detecting Online Game Bots

The abuse of online games by automated programs, known as game bots, for gaining unfair advantages has plagued millions of participating players with escalating severity in recent years. The current methods for distinguishing bots and humans are based on human interactive proofs (HIPs), such as CAPTCHAs. However, HIP-based approaches have inherent drawbacks. In particular, they are too obtrusive to be tolerated by human players in a gaming context. In this chapter, we propose a non-interactive approach based on human observational proofs (HOPs) for continuous game bot detection. HOPs differentiate bots from human players by passively monitoring input actions that are difficult for current bots to perform in a human-like manner. We collect a series of user-input traces in one of the most popular online games, World of Warcraft. Based on the traces, we characterize the game playing behaviors of bots and humans. Then, we develop a HOP-based game bot defense system that analyzes user-input actions with a cascade-correlation neural network to distinguish bots from humans. The HOP system is effective in capturing current game bots, which raises the bar against game exploits and forces a determined adversary to build more complicated game bots for detection evasion in the future.

The remainder of this dissertation is organized as follows. Chapter 2 covers modeling covert timing channels in terms of capacity and introduces model-based covert timing channels. Chapter 3 describes other covert timing channels and related detection methods, and details our entropy-based approach to detection. Chapter 4 details our chat bot measurements and our hybrid classification system. Chapter 5 describes our game bot measurements and our HOP-based game bot defense system. Lastly, in Chapter 6 we conclude and outline possible directions for our future work.

Chapter 2

Designing and Modeling Covert Timing Channels

A covert channel is a "communication channel that can be exploited by a process to transfer information in a manner that violates a system's security policy" [31]. There are two types of covert channels: covert storage channels and covert timing channels. A covert storage channel manipulates the contents of a storage location (e.g., disk, memory, packet headers, etc.) to transfer information. A covert timing channel manipulates the timing or ordering of events (e.g., disk accesses, memory accesses, packet arrivals, etc.) to transfer information. The focus of this chapter is on covert timing channels.

The potential damage of a covert timing channel is measured in terms of its capacity. The capacity of covert timing channels has been increasing with the development of high-performance computers and high-speed networks. While covert timing channels studied in the 1970s could transfer only a few bits per second [70], covert timing channels in modern computers can transfer several megabits per second [125]. To defend against covert timing channels, researchers have proposed various methods to detect and disrupt them. The disruption of covert timing channels manipulates traffic to slow or stop covert timing channels [42, 62, 61, 63, 48]. The detection of covert timing channels mainly uses statistical tests to differentiate covert traffic from legitimate traffic [12, 13, 19, 110, 44]. Such detection methods are somewhat successful, because most existing covert timing channels cause large deviations in the timing behavior from that of normal traffic, making them relatively easy to detect.

In this chapter, we introduce model-based covert timing channels, which endeavor to evade detection by modeling and mimicking the statistical properties of legitimate traffic. We design and develop a framework for building model-based covert timing channels, in which hidden information is carried through pseudo-random values generated from a distribution function. We use the inverse distribution function and cumulative distribution function for encoding and decoding. The framework includes four components, filter, analyzer, encoder, and transmitter. The filter profiles the legitimate traffic, and the analyzer fits the legitimate traffic behavior to a model. Then, based on the model, the encoder chooses the appropriate distribution functions from statistical tools and traffic generation libraries to create covert timing channels. The distribution functions and their parameters are determined by automated model fitting. The process of model fitting proves very efficient and the induced overhead is minor. Lastly, the transmitter generates covert traffic and blends with legitimate traffic.

The two primary design goals of covert timing channels are high capacity and detection resistance. To evaluate the effectiveness of the proposed framework, we perform a series of LAN and WAN experiments to measure the capacity and detection resistance of our model-based covert timing channel. We estimate the capacity with a model and then validate the model with real experiments. Our experimental results show that the capacity is close to that of an optimal covert timing channel that transmits in a similar condition. In previous research, it is shown that the shape [12, 13] and regularity [19, 110] of network traffic are important properties in the detection of covert timing channels. We evaluate the detection resistance of the proposed framework using shape and regularity tests. The experimental results show that both tests fail to differentiate the model-based covert traffic from legitimate traffic. Overall, our model-based covert timing channel achieves strong detection resistance and high capacity.

There is an arms race between covert timing channel design and detection. To maintain the lead, researchers need to continue to improve detection methods and investigate new attacks. The goal of this work is to increase the understanding of more advanced covert timing channel design. Our demonstration of model-based covert timing channels motivates the development of a more advanced detection method based on entropy, which is discussed in Chapter 3.

2.1 Background

In this section, we describe basic communication concepts and relate them to covert timing channels. Then, based on these concepts, we formulate two base cases in covert timing channel design. The basic problem of communication, producing a message at one point and reproducing that message at another point, is the same for both overt and covert channels, although covert channels must consider the additional problem of hiding communication.

2.1.1 Basic Communication Concepts

The capacity of a communication channel is the maximum rate that it can reliably transmit information. The capacity of a covert timing channel is measured in bits per time unit [88]. The capacity in bits per time unit C_t is defined as:

$$C_t = \max_X \frac{I(X;Y)}{E(X)},$$

where X is the transmitted inter-packet delays or input distribution, Y is the received inter-packet delays or output distribution, $I(X;Y)$ is the mutual information between X and Y, and $E(X)$ is the expected time of X.

The mutual information measures how much information is carried across the channel from X to Y. The mutual information $I(X;Y)$ is defined as:

$$I(X;Y) = \begin{cases} \sum\limits_X \sum\limits_Y P(y \mid x)P(x)\log\frac{P(y|x)P(x)}{P(x)P(y)}, \text{(discrete)} \\ \int\limits_X \int\limits_Y P(y \mid x)P(x)\log\frac{P(y|x)P(x)}{P(x)P(y)}\, dx\, dy, \text{(continuous)} \end{cases}$$

The noise, represented by the conditional probability in the above definitions, is defined as:

$$P(y \mid x) = f_{noise}(y, x),$$

where f_{noise} is the noise probability density function, x is the transmitted inter-packet delays, and y is the received inter-packet delays.

The noise distribution f_{noise} is the probability that the transmitted inter-packet delay x results in the received inter-packet delay y. The specific noise distribution for inter-packet delays is detailed in Section 2.3.2.

2.1.2 Base Cases in Design

The two main goals of covert timing channel design are high capacity and detection resistance. There are few examples of practical implementations of covert timing channels in the literature, so we begin to explore the design space in terms of both capacity and detection resistance. The focus of our model-based covert timing channel is to achieve high detection resistance. In the following section, we formulate two base cases in covert channel design as comparison to the model-based covert timing channel.

The first case, optimal capacity, transmits as much information as possible, sending hundreds or more packets per second. Such a design might not be able to achieve covert communication, but is useful as a theoretical upper bound. The second case, fixed average packet rate, sends packets at a specific fixed average packet rate, encoding as much information per packet as possible. The fixed average packet rate is mainly determined by the packet rate of legitimate traffic.

2.1.2.1 Optimal Capacity Channel

The first design, OPtimal Capacity (OPC), uses the discrete input distribution that transmits information as fast as possible. The optimal capacity is dependent on the optimal distance between two symbols. The first symbol is (approximately) zero and the second symbol is non-zero, so the use of more symbols (i.e., four or eight) will introduce more non-zero symbols and decrease the symbol rate. The use of smaller distances between the two symbols increases the symbol rate and the error rate. The optimal distance is the point at which the increase in error rate balances the increase in symbol rate.

The code operates based on two functions. The encode function is defined as:

$$F_{encode}(s) = d_s = \begin{cases} 0, & s = 0 \\ d, & s = 1 \end{cases}$$

where s is a symbol, d_s is an inter-packet delay with a hidden symbol s, and d is the optimal distance between the two symbols. The decode function is defined as:

$$F_{decode}(d_s) = s = \begin{cases} 0, & d_s < \frac{1}{2}d \\ 1, & \frac{1}{2}d \le d_s \end{cases}$$

where d_s is an inter-packet delay with a hidden symbol s.

Channel Capacity: The channel capacity of OPC is dependent on the optimal input distribution and noise. The input distribution is defined as:

$$P(x) = \begin{cases} p, & x = d \\ 1 - p, & x = 0 \\ 0, & \text{otherwise} \end{cases}$$

where p is the probability of the symbol $s = 1$, and $1 - p$ is the probability of the symbol $s = 0$.

Therefore, the capacity of OPC is the maximum of the mutual information with respect to the parameters d and p of the input distribution over the expected time $d \cdot p$:

$$C_t = \max_{d,p} \frac{1}{d \cdot p} \sum_X \sum_Y P(y \mid x)P(x)\log\frac{P(y \mid x)P(x)}{P(x)P(y)}.$$

2.1.2.2 Fixed-Average Packet Rate Channel

The second design, Fixed-average Packet Rate (FPR), uses the input distribution that encodes as much information per packet as possible with a constraint on the average cost of symbols. The cost is measured in terms of the time required for symbol transmission. Therefore, the optimal input distribution is subject to the constraint on the average packet rate, i.e., the cost of symbol transmission.

The optimal input distribution for FPR is computed with the Arimoto-Blahut algorithm generalized for cost constraints [14]. The Arimoto-Blahut algorithm computes the optimal input distribution for capacity in bits per channel usage. The capacity in bits per channel usage C_u is defined as:

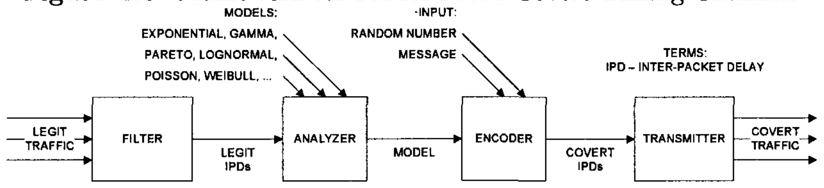
$$C_u = \max_X I(X;Y).$$

In general, C_u and C_t do not have the same input distribution X. However, if the input distribution is constrained so that $E(X) = c$ (where c is a constant), then the optimal input distribution X is optimal for both C_u and C_t, and $C_u = C_t \cdot c$. Thus, FPR transmits as much information per packet (channel usage) and per second (time unit) as possible with a fixed average packet rate. We use the Arimoto-Blahut algorithm to compute the optimal input distribution for FPR. The capacity results for FPR, based on the Arimoto-Blahut algorithm, are detailed in Section 2.3.

2.2 The Framework

The covert timing channel framework, as shown in Figure 2.1, is a pipeline that filters and analyzes legitimate traffic then encodes and transmits covert traffic. As the output of the pipeline, the covert traffic mimics the observed legitimate traffic, making it easy to evade detection. The components of the framework include filter, analyzer, encoder, and transmitter, which are detailed in the following paragraphs.

Figure 2.1: Framework for Model-Based Covert Timing Channels.



The filter monitors the background traffic and singles out the specific type of traffic to be mimicked. The more specific application traffic the filter can identify and profile, the better model we can have for generating covert traffic. For example, FTP is an application protocol based on TCP, but generating a series of inter-packet delays based on a model of all TCP traffic would be a poor model for describing FTP behaviors. Once the specified traffic is filtered, the traffic is further classified into individual flows based on source and destination IP addresses. The filter then calculates the inter-packet delay between subsequent pair of packets from each flow, and forwards the results to the analyzer.

The analyzer fits the inter-packet delays in sets of 100 packets with the Exponential, Gamma, Pareto, Lognormal, Poisson, and Weibull distributions. The fitting process uses maximum likelihood estimation (MLE) to determine the parameters for each model. The model with the smallest root mean squared error (RMSE), which measures the difference between the model and the estimated distribution, is chosen as the traffic model. The model selection is automated. Other than the set of models provided to the analyzer, there is no human input. The models are scored based on root mean squared errors, as shown in Table 2.1. The model with the lowest root mean squared error is the closest to the data being modeled. Since most types of network traffic are non-stationary [22], the analyzer supports piecewise modeling of non-stationary processes by adjusting the parameters of the model after each set of 100 covert inter-packet delays. The analyzer refits the current model with new sets of 100 packets to adjust the parameters. The analyzer can take advantage of a larger selection of models to more accurately model different types of application traffic. For

Table 2.1: Scores for Models of HTTP Inter-Packet Delays

model	parameters	root mean squared error
Weibull	0.0794, 0.2627	0.0032
Gamma	0.1167, 100.8180	0.0063
Lognormal	-4.3589, 3.5359	0.0063
Pareto	3.6751, 0.0018	0.0150
Poisson	11.7611	0.0226
Exponential	11.7611	0.0294

example, if we know that the targeted traffic is well-modeled as an Erlang distribution, we will add this distribution to the set of models. For each of the current models, the computational overhead is less than 0.1 milliseconds and the storage overhead for the executable is less than 500 bytes, so the induced resource consumption for supporting additional models is not an issue.

The filter and analyzer can be run either offline or online. In the offline mode, the selection of the model and parameters is based on traffic samples. The offline mode consumes less resources, but the model might not represent the current network traffic behavior well. In the online mode, the selection of the model and parameters is based on live traffic. The online mode consumes more resources and requires that the model and parameters be transmitted to the decoder with the support of a startup protocol, but the model better represents the current network traffic behavior. The startup protocol is a model determined in advance, and is used to transmit the online model (1 byte) and parameters (4-8 bytes) to the decoder.

The encoder generates random covert inter-packet delays that mimic legitimate inter-packet delays. The input to the encoder includes the model, the message, and a sequence of random numbers. Its output is a sequence of covert random inter-packet delays. The message to be sent is separated into symbols. The symbols map to different random timing values based on a random code that distributes symbols based on the model.

Using a sequence of random numbers r_1, r_2, ..., r_n., we transform the discrete symbols into continuous ones. The continuization function is

$$F_{continuize}(s) = (\frac{s}{\mid S \mid} + r) \bmod 1 = r_s,$$

where S is the set of possible symbols, s is a symbol and r is a Uniform(0,1) random variable. The corresponding discretization function is:

$$F_{discretize}(r_s) = \mid S \mid \cdot ((r_s - r) \bmod 1) = s,$$

where r_s is a Uniform(0,1) random variable with a hidden symbol s.

The encoder and decoder start with the same seed and generate the same sequence of random numbers, r_1, r_2, ..., r_n. To maintain synchronization, the encoder and decoder associate the sequence of symbols with TCP sequence numbers, i.e., s_1 with the first TCP sequence number, s_2 with the second TCP sequence number, and so on. [1] Therefore, both the encoder and decoder have the same values of r through the sequence of symbols. The inverse distribution function F^{-1}_{model} takes a Uniform(0,1) random number as input and generates a random variable from the selected model

[1]With this mechanism, repacketization can cause synchronization problems, so other mechanisms such as "bit stuffing" [110] could be useful for synchronization.

as output. The sequence of transformed random numbers r_{s1}, r_{s2}, ..., r_{sn} is used with the inverse distribution function to create random covert inter-packet delays d_{s1}, d_{s2}, ..., d_{sn}. The encode function is:

$$F_{encode} = F^{-1}_{model}(r_s) = d_s,$$

where F^{-1}_{model} is the inverse distribution function of the selected model. The decode function is:

$$F_{decode} = F_{model}(d_s) = r_s,$$

where F_{model} is the cumulative distribution function of the selected model, and d_s is a random covert inter-packet delay with a hidden symbol s.

The transmitter sends out packets to produce the random covert inter-packet delays d_{s1}, d_{s2}, ..., d_{sn}. The receiver then decodes and discretizes them to recover the original symbols s_1, s_2, ..., s_n.

2.2.1 Model-Based Channel Capacity

The model-based channel capacity is also dependent on the input distribution and noise. The input distribution is defined as:

$$P(x) = f_{model}(x)$$

where f_{model} is the probability density function of the selected model.

Therefore, the capacity of the model-based channel is the mutual information over the expected time $E(X)$:

$$C_t = \frac{1}{E(X)} \int\limits_X \int\limits_Y P(y \mid x)P(x)\log\frac{P(y \mid x)P(x)}{P(x)P(y)}.$$

2.2.2 Implementation Details

We implement the proposed framework using `C` and `MATLAB` in Unix/Linux environments. The components run as user-space processes, while access to `tcpdump` is required. The filter is written in C and runs `tcpdump` with a user-specified filtering expression to read the stream of packets. The filter processes the traffic stream and computes the inter-packet delays based on the packet timestamps. The analyzer is written in `MATLAB` and utilizes the fitting functions from the statistics toolbox for maximum likelihood estimation.

The encoder is written in `C`, and uses random number generation and random variable models from the Park-Leemis [71] simulation C libraries. The transmitter is also written in `C`, with some inline assembly, and uses the Socket API. The timing mechanism used is the Pentium CPU Time-Stamp Counter, which is accessed by calling the `RDTSC` (Read Time-Stamp Counter) instruction. The `RDTSC` instruction has excellent resolution and low overhead, but must be calibrated to be used as a general purpose timing mechanism. The `usleep` and `nanosleep` functions force a context switch, which delays the packet transmission with small inter-packet delays, so these functions are not used.

2.3 Experimental Evaluation

In this section, we evaluate the effectiveness of a model-based covert timing channel built from our framework. The OPC and FPR covert timing channels, discussed in Section 2.1, are used as points of comparison. In particular, we examine the capacity and detection resistance of each covert timing channel.

2.3.1 Experimental Setup

The defensive perimeter of a network, composed of firewalls and intrusion detection systems, is responsible for protecting the network. Typically, only a few specific application protocols, such as HTTP and SMTP, are commonly allowed to pass through the defensive perimeter. We utilize outgoing HTTP inter-packet delays as the medium to build model-based covert timing channels, due to the wide acceptance of HTTP traffic for crossing the network perimeter. We refer to the model-based HTTP covert timing channel as MB-HTTP.

2.3.1.1 Testing Scenarios

There are three different testing scenarios in our experimental evaluation. The first scenario is in a LAN environment, a medium-size campus network with subnets for administration, departments, and residences. The LAN connection is between two machines, located in different subnets. The connection passes through several switches, the routers inside the campus network, and a firewall device that protects each subnet. The other two scenarios are in WAN environments. The first WAN connection is between two machines, both are on the east coast of the United States but in different states. One is on a residential cable network and the other is on a medium-size campus network. The second WAN connection is between two machines on the opposite coasts of the United States, one on the east coast and the other on the west coast. Both machines are on campus networks.

Table 2.2: Network Conditions for Test Scenarios

	LAN	WAN E-E	WAN E-W
distance	0.3 miles	525 miles	2660 miles
RTT	1.766ms	59.647ms	87.236ms
IPDV	2.5822e-05	2.4124e-03	2.1771e-04
hops	3	18	13
IPDV - inter-packet delay variation			

The network conditions for different experiment scenarios are summarized in Table 2.2. The two-way round-trip time (RTT) is measured using the ping command. We compute the one-way inter-packet delay variation based on the delays between packets leaving the source and arriving at the destination. The inter-packet delay variations of the three connections span three orders of magnitude, from 1×10^{-3} to 1×10^{-5}. The LAN connection has the lowest inter-packet delay variation and the two WAN connections have higher inter-packet delay variation, as expected. The WAN E-E connection is shorter and has smaller RTT time than the WAN E-W connection. However, WAN E-E has higher inter-packet delay variation than WAN E-W, due to more traversed hops. This implies that the inter-packet delays variation is more sensitive to the number of hops than the physical distance and RTT between two machines.

2.3.1.2 Building MB-HTTP

We install the components of the framework on the testing machines. The filter distinguishes the outgoing HTTP traffic from background traffic. The analyzer observes 10 million HTTP inter-packet delays, then fits the HTTP inter-packet delays to the models, as described in Section 2.2. The fitting functions use maximum likelihood estimation (MLE) to determine the parameters for each model. The model with the best root mean squared error (RMSE), a measure of the difference between the model and the distribution being estimated, is chosen as the traffic model.

For the HTTP inter-packet delays, the analyzer selects the Weibull distribution based on the root mean squared error. Note that HTTP inter-packet delays have been shown to be well approximated by a Weibull distribution [22]. The Weibull probability distribution function is:

$$f(x, \lambda, k) = \frac{k}{\lambda}(\frac{x}{\lambda})^{(k-1)} e^{-(\frac{x}{\lambda})^k}.$$

The parameters, which vary for each set of 100 packets, have a mean scale parameter λ of 0.0371 and a mean shape parameter k of 0.3010. With these parameters, the mean inter-packet delay is 0.3385, approximately 3 packets per second.

2.3.1.3 Formulating OPC and FPR

The average packet rate for FPR is fixed at $\frac{1}{0.3385} = 2.954$ packets per second, based on the average packet rate of HTTP traffic. We use the Arimoto-Blahut algorithm to compute the optimal input distribution, with the average packet rate of 2.954 as

Table 2.3: Mean Packets/Second and Inter-Packet Delay for OPC

	LAN		WAN E-E		WAN E-W	
channel	PPS	IPD	PPS	IPD	PPS	IPD
OPC	12,777.98	7.87e-05	137.48	7.31e-03	1,515.56	6.63e-04
PPS - mean packets per second, IPD - mean inter-packet delay						

the cost constraint. The optimal input distribution balances high cost symbols with low probabilities and low cost symbols with high probabilities, such that the average cost constraint is satisfied. The constraint can be satisfied for infinitely large symbols with infinitely small probabilities, and hence, the optimal input distribution decays exponentially to infinity. The results of the Arimoto-Blahut algorithm, as the number of intervals increases, reduce to an Exponential distribution with an inverse scale parameter of $\lambda = 2.954$. The Exponential probability distribution function is:

$$f(x, \lambda) = \lambda e^{-\lambda x}.$$

We compute the optimal distance between packets for OPC based on the noise distribution. The optimal distance between packets and the average packet rate for OPC is shown in Table 2.3. For connections with higher inter-packet delay variation, OPC increases the time elapse between packets to make the inter-packet delays easier to distinguish, and, as a result, lowers the average number of packets per second.

2.3.2 Capacity

The definition of capacity allows us to estimate the capacity of each covert timing channel based on the network conditions of each connection. In previous research [138], the inter-packet delay differences have been shown to be well-modeled by a Laplace distribution. The probability density function of the Laplace distribution is:

$$f(x, \mu, b) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}.$$

The setting of the scale parameter b is based on the inter-packet delay variation for each connection. The variation of the Laplace distribution is $\sigma^2 = 2b^2$. Therefore, we set b to:

$$b = \sqrt{\frac{1}{2}\sigma^2},$$

where σ^2 is the inter-packet delay variation for each connection.

Table 2.4: Theoretical Capacity for Covert Timing Channels

	LAN		WAN E-E		WAN E-W	
channel	CPP	CPS	CPP	CPS	CPP	CPS
MB-HTTP	9.39	27.76	4.12	12.19	6.84	20.21
FPR	12.63	37.32	6.15	18.17	9.59	28.35
OPC	0.50	6395.39	0.50	68.80	0.50	758.54
CPP - capacity per packet, CPS - capacity per second						

The results, in terms of capacity per packet and capacity per second, are shown in Table 2.4. While OPC has the highest capacity, it is the least efficient in terms of capacity per packet. Furthermore, with the large number of packets per second, it can be easily detected by most intrusion detection systems.

The capacity of MB-HTTP is 67% to 74% of that of FPR, with larger differences for connections with high inter-packet delay variation than for those with low inter-packet delay variation. The Weibull distribution has a larger proportion of very small values than the Exponential distribution. As a result, MB-HTTP uses more small values than FPR and benefits more from lower inter-packet delay variation.

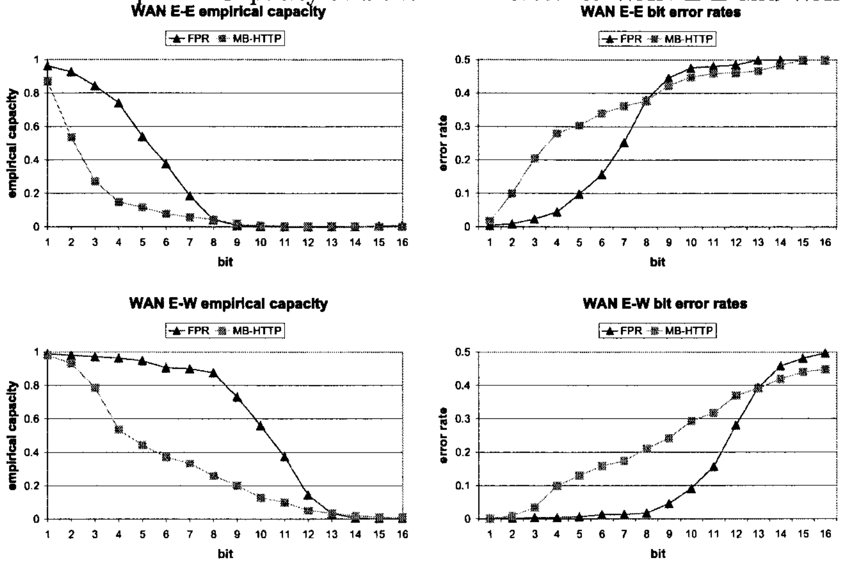
The theoretical capacity is somewhat optimistic. The model only considers the noise introduced after packets leave the transmitter. With the real covert timing channels, noise is introduced before packets leave the transmitter. The transmitter is sometimes not able to transmit at the appropriate times, due to slow processing, context switches, etc. Thus, the actual distance between packets can increase or decrease from the intended distance as the packets are transmitted.

2.3.2.1 Empirical Capacity

To evaluate the channel capacity in practice, we run covert timing channels on each connection. The channels are configured to transmit 16,000 random bits of information. For FPR and MB-HTTP, the number of bits encoded per packet is set to 16 (i.e., $2^{16} = 65,536$ different values), while OPC transmits a single bit per packet.

During these tests, we measure the bit error rate of each covert timing channel from the most significant bit to the least significant bit of each packet. The most significant bit represents a large part of the inter-packet delay, where the least significant bit represents a small part of the inter-packet delay. While flipping the most significant bit causes a difference in seconds or tenths of seconds, changing the least significant bit means a difference only in milliseconds or microseconds. In other words, the higher the number of bits encoded per packet, the smaller the precision of the lowest order bits. Interestingly, encoding at 16 bits per packet and decoding at 8 bits per packet

Figure 2.2: Empirical Capacity and Bit Error Rates for WAN E-E and WAN E-W



produces the most significant 8 bits of the 16 bit code.

To determine the transmission rate with error correction, we measure the empirical capacity of each bit as a binary symmetric channel. The binary symmetric channel is a special case where the channel has two symbols of equal probability. The capacity of a binary symmetric channel is:

$$C = I(X;Y) = 1 - (p \log p + q \log q),$$

where p is the probability of a correct bit and $q = 1 - p$ is the probability of an incorrect bit.

The empirical capacity and bit error rate for each bit, from the most significant to the least significant, are shown in Figure 2.2. The empirical capacity per bit degrades as the bit error rates increase. The total capacity of the channel is the summation of the capacity for each bit. For MB-HTTP, the bit error rate increases somewhat linearly. For FPR, the bit error rate accelerates gradually, eventually overtaking the bit error rates of MB-HTTP, though at this point the capacity per bit is insignificant.

Table 2.5: Empirical Capacity for Covert Timing Channels

	LAN		WAN E-E		WAN E-W	
channel	ECPP	ECPS	ECPP	ECPS	ECPP	ECPS
MB-HTTP	6.74	19.93	2.15	6.35	5.18	15.31
FPR	10.95	32.35	4.63	13.67	9.37	27.69
OPC	0.85	10,899.62	0.66	91.28	0.98	1,512.53
ECPP - empirical capacity per packet, ECPS - empirical capacity per second						

The empirical capacity of each covert timing channel is shown in Table 2.5. The empirical capacity of MB-HTTP is still about 46% to 61% of that of FPR, somewhat lower than the case in the theoretical model. This is because a larger proportion of MB-HTTP traffic has small inter-packet delays than that of FPR, and small inter-packet delays are more sensitive to noise caused by transmission delays (i.e., slow processing, context switches, etc.) than large inter-packet delays, which is not represented in the theoretical model.

2.3.3 Detection Resistance

The detection resistance, as described in Section 2.1, is estimated based on the shape and regularity tests. To examine the shape of the distribution, we use the Kolmogorov-Smirnov test [36], which is a non-parametric goodness-of-fit test. To examine the regularity of the traffic, we use the regularity test [19], which studies the variance of the traffic pattern. In this section, we detail these two tests and show the detection resistance of MB-HTTP against both tests.

2.3.3.1 Shape Tests

The two-sample Kolmogorov-Smirnov test determines whether or not two samples come from the same distribution. The Kolmogorov-Smirnov test is distribution free, meaning the test is not dependent on a specific distribution. Thus, it is applicable to a variety of types of traffic with different distributions. The Kolmogorov-Smirnov test statistic measures the maximum distance between two empirical distribution functions.

$$KSTEST = \max \mid S_1(x) - S_2(x) \mid,$$

where S_1 and S_2 are the empirical distribution functions of the two samples.

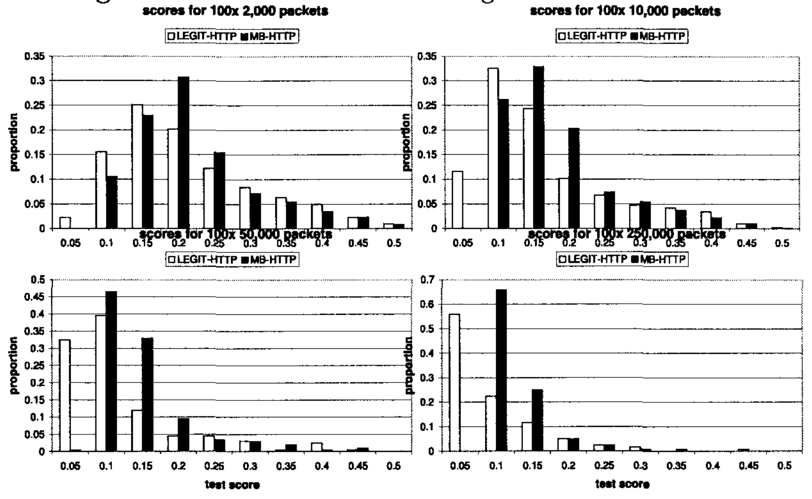
In our experiments, we test a large set of legitimate inter-packet delays against a sample of either covert or legitimate inter-packet delays. The large set is a training set of 10,000,000 HTTP inter-packet delays. The training set is used to represent the normal behavior of the HTTP protocol.

The test score by comparing the two sets is used to determine if the sample is covert or legitimate. A small score indicates that the behavior is close to normal. However, if the test score is large, i.e., the sample does not fit the normal behavior of the protocol, it indicates a potential covert timing channel.

Table 2.6: Mean and Standard Deviation of Kolmogorov-Smirnov Test Scores

	LEGIT-HTTP		MB-HTTP		FPR		OPC	
sample size	mean	stdev	mean	stdev	mean	stdev	mean	stdev
100x 2,000	.193	.110	.196	.093	.925	.002	.999	.000
100x 10,000	.141	.103	.157	.087	.925	.001	.999	.000
100x 50,000	.096	.088	.122	.073	.924	.000	.999	.000
100x 250,000	.069	.066	.096	.036	.924	.000	.999	.000

Figure 2.3: Distribution of Kolmogorov-Smirnov Test Scores



The Kolmogorov-Smirnov test is run 100 times for each of 2,000, 10,000, 50,000, and 250,000 packet samples of legitimate and covert traffic from each covert timing channel. The mean and standard deviation of the test scores are shown in Table 2.6. For FPR and OPC, the mean scores are over 0.90 and the standard deviations are extremely low, indicating that the test can reliably differentiate both covert timing channels from normal HTTP traffic. By contrast, the mean scores for MB-HTTP samples are very close to those of legitimate samples. The mean scores are for 100 tests, which in total include as many as 25 million (250,000 x 100) inter-packet delays. The distribution of individual test scores is illustrated in Figure 2.3.

Table 2.7: False Positive and True Positive Rates for Kolmogorov-Smirnov Test

		LEGIT	MB-HTTP	FPR	OPC
sample size	cutoff	false pos.	true pos.	true pos.	true pos.
100x 2,000	$KSTEST \geq .66$	.01	.01	1.00	1.00
100x 10,000	$KSTEST \geq .65$	.01	.01	1.00	1.00
100x 50,000	$KSTEST \geq .41$	.01	.01	1.00	1.00
100x 250,000	$KSTEST \geq .21$	.01	.02	1.00	1.00

The detection resistance based on the Kolmogorov-Smirnov test is shown in Table 2.7. The targeted false positive rate is 0.01. To achieve this false positive rate, the cutoff scores—the scores that decide whether samples are legitimate or covert—are set at the 99th percentile of legitimate sample scores. The true positive rates, based on the cutoff scores, are then shown for each covert timing channel. Since the true positive rates in all 100 tests are 1.00, the Kolmogorov-Smirnov test detects FPR and OPC easily. However, the true positive rates for MB-HTTP are approximately the same as the false positive rates. The Kolmogorov-Smirnov test cannot differentiate between MB-HTTP and legitimate samples. Such a result can be explained based on the distribution of individual test scores, which is shown in Figure 2.3. While the mean scores of MB-HTTP traffic in Table 2.6 are slightly higher than those of LEGIT-HTTP, the distributions of individual scores overlap so that the false positive rate and true positive rate are approximately equal.

2.3.3.2 Regularity Tests

The regularity test [19] determines whether the variance of the inter-packet delays is relatively constant or not. This test is based on the observation that for most types of network traffic, the variance of the inter-packet delays changes over time. With covert timing channels, the code used to transmit data is a regular process and, as a result, the variance of the inter-packet delays remains relatively constant over time.

In our experiments, we test the regularity of a sample of either covert or legitimate inter-packet delays. The sample is separated into sets of w inter-packet delays. Then, for each set, the standard deviation of the set σ_i is computed. The regularity is the standard deviation of the pairwise differences between each σ_i and σ_j for all sets $i < j$.

$$regularity = STDEV(\frac{| \sigma_i - \sigma_j |}{\sigma_i}, i < j, \forall i, j)$$

The regularity test is run 100 times for 2,000 packet samples of legitimate and covert samples from each covert timing channel. The window sizes of $w = 100$ and $w = 250$ are used. The mean regularity scores are shown in Table 2.8. If the regularity is small, the sample is highly regular, indicating a potential covert timing channel.

Table 2.8: Mean of Regularity Test Scores

sample size	LEGIT-HTTP	MB-HTTP	FPR	OPC
100x 2,000 w=100	43.80	38.21	0.34	0.00
100x 2,000 w=250	23.74	22.87	0.26	0.00

The mean regularity scores for OPC are 0.0 for both tests, indicating regular behavior. There are two values, each with 0.5 probability. Therefore, the standard deviation within sets is small $\sigma = 0.5d = 3.317e-4$, and there is no detectable change in the standard deviation between sets. The mean regularity score for FPR is small as well, showing that the test is able to detect the regular behavior. While the standard deviation of FPR, which is based on the Exponential distribution, is $\sigma = \lambda = 0.3385$, the code is a regular process, so the variance of the inter-packet delays remains relatively constant.

The mean regularity scores for MB-HTTP are close to those of legitimate samples. This is because the parameters are recalibrated after each set of 100 packets, as described in Section 2.2. The parameters of the distribution determine the mean and standard deviation, so adjusting the parameters changes the variance after each set of 100 inter-packet delays. As a result, like legitimate traffic, the variance of the inter-packet delays appears irregular.

Table 2.9: False Positive and True Positive Rates for Regularity Test

		LEGIT	MB-HTTP	FPR	OPC
sample size	cutoff	false pos.	true pos.	true pos.	true pos.
100x 2,000 w=100	$reg. \leq 6.90$	.01	.00	1.00	1.00
100x 2,000 w=250	$reg. \leq 5.20$	.01	.00	1.00	1.00

The detection resistance based on the regularity test is shown in Table 2.9. The targeted false positive rate is 0.01. The cutoff scores are set at the 1st percentile of legitimate sample scores, in order to achieve this false positive rate. The true positive rates, based on the cutoff scores, are then shown for each covert timing channels. The regularity test is able to detect FPR and OPC in all 100 tests. The resulting true positive rates for MB-HTTP are approximately the same as the false positive rate. Basically, the test is no better than random guessing at detecting MB-HTTP.

2.4 Conclusion

We introduced model-based covert timing channels, which mimic the observed behavior of legitimate network traffic to evade detection. We presented a framework for building such model-based covert timing channels. The framework consists of four components: filter, analyzer, encoder, and transmitter. The filter characterizes the specific features of legitimate traffic that are of interest. The analyzer fits the traffic to several models and selects the model with the best fit. The encoder generates random covert inter-packet delays that, based on the model, mimic the legitimate traffic. The transmitter then manipulates the timing of packets to create the model-based covert timing channel.

Using channel capacity and detection resistance as major metrics, we evaluated the proposed framework in both LAN and WAN environments. Our capacity results suggest that model-based covert timing channels work efficiently even in the coast-to-coast scenario. Our detection resistance results show that, for both shape and regularity tests, covert traffic is sufficiently similar to legitimate traffic that current detection methods cannot differentiate them. In contrast, the Kolmogorov-Smirnov and regularity tests easily detect FPR and OPC.

Chapter 3

Detecting Covert Timing Channels

3.1 Introduction

As an effective way to exfiltrate data from a well-protected network, a covert timing channel manipulates the timing or ordering of network events (e.g., packet arrivals) for secret information transfer over the Internet, even without compromising an end-host inside the network. On the one hand, such information leakage caused by a covert timing channel poses a serious threat to Internet users. Their secret credentials like passwords and keys could be stolen through a covert timing channel without much difficulty. On the other hand, detecting covert timing channels is a well-known challenging task in the security community.

In general, the detection of covert timing channels uses statistical tests to differentiate covert traffic from legitimate traffic. However, due to the high variation in legitimate network traffic, detection methods based on standard statistical tests are not accurate and robust in capturing a covert timing channel. Although there have been recent research efforts on detecting covert timing channels over the Internet [18, 19, 110, 75, 45], some detection methods are designed to target one specific covert timing channel and therefore fail to detect other types of covert timing channels; the other detection methods are broader in detection but are over-sensitive to the high variation of network traffic. In short, none of the previous detection methods are effective at detecting a variety of covert timing channels.

In this chapter, we propose a new entropy-based approach to detecting covert timing channels. The entropy of a process is a measure of uncertainty or information content, a concept that is of great importance in information and communication theory [111]. While entropy has been used in covert timing channel capacity analysis, it has never been used to detect covert timing channels. We observe that a covert timing channel cannot be created without causing some effects on the entropy of the original process[1]. Therefore, a change in the entropy of a process provides a critical clue for covert timing channel detection.

More specifically, we investigate the use of entropy and conditional entropy in detecting covert timing channels. For finite samples, the exact entropy rate of a process cannot be measured and must be estimated. Thus, we estimate the entropy rate with the corrected conditional entropy, a measure used on biological processes [101]. The corrected conditional entropy is designed to be accurate with limited data,

[1]This observation applies to complex processes, like network traffic, but not to simple independent and identically distributed processes [20].

which makes it excellent for small samples of network data. To evaluate our new entropy-based approach, we conduct a series of experiments to validate whether our approach is capable of differentiating covert traffic from legitimate traffic. We perform the fine-binned estimation of entropy and the coarse-binned estimation of corrected conditional entropy for both covert and legitimate samples. We then determine false positive and true positive rates for both types of estimations. Our experimental results show that the combination of entropy and corrected conditional entropy is very effective in detecting covert timing channels.

3.2 Background And Related Work

To defend against covert timing channels, researchers have proposed different solutions to detect, disrupt, and eliminate covert traffic. The disruption of covert timing channels adds random delays to traffic, which reduces the capacity of covert timing channels but degrades system performance as well. The detection of covert timing channels is accomplished using statistical tests to differentiate covert traffic from legitimate traffic. While the focus of earlier work is on disrupting covert timing channels [48, 58, 62, 61, 63] or on eliminating them in the design of systems [1, 65, 66], more recent research has begun to investigate the design and detection of covert timing channels [18, 19, 110, 75, 76, 45, 109]. In the following subsections, we give an overview of recent research on covert timing channels and detection tests.

3.2.1 Covert Timing Channels

There are two types of covert timing channels: active and passive. In terms of covert timing channels, active refers to covert timing channels that generate additional traffic to transmit information, while passive refers to covert timing channels that manipulate the timing of existing traffic. In general, active covert timing channels are faster, but passive covert timing channels are more difficult to detect. On the other hand, active covert timing channels often require a compromised machine, whereas passive covert timing channels, if creatively positioned, do not. The majority of the covert timing channels discussed in this section are active covert timing channels, except where stated otherwise.

Figure 3.1: IPCTC/TRCTC/MBCTC Scenario

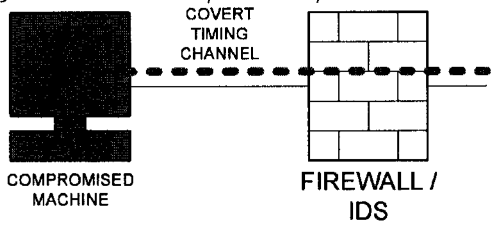
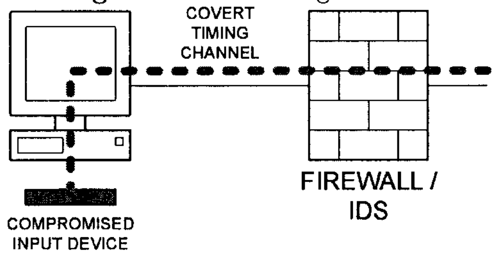


Figure 3.2: JitterBug Scenario



3.2.1.1 IP Covert Timing Channel

Cabuk et al. [19] developed the first IP covert timing channel, which we refer to as IPCTC, and investigated a number of design issues. A scenario where IPCTC can be used is illustrated in Figure 3.1. In this scenario, a machine is compromised, and the defensive perimeter, represented as a perimeter firewall or intrusion detection system, monitors communication with the outside. Therefore, a covert timing channel can be used to pass through the defensive perimeter undetected. IPCTC uses a simple interval-based encoding scheme to transmit information. IPCTC transmits a 1-bit by sending a packet during an interval and transmits a 0-bit by not sending a packet during an interval. A major advantage to this scheme is that when a packet is lost, a bit is flipped but synchronization is not affected. The timing-interval t and the number of 0-bits between two 1-bits determines the distribution of IPCTC inter-packet delays. It is interesting to note that if the pattern of bits is uniform, the distribution of inter-packet delays is close to a Geometric distribution. To avoid creating a pattern of inter-packet delays at multiples of a single t, the timing-interval t is rotated among different values.

3.2.1.2 Time-Replay Covert Timing Channel

Cabuk [18] later designed a more advanced covert timing channel based on a replay attack, which we refer to as TRCTC. TRCTC uses a sample of legitimate traffic S_{in} as input and replays S_{in} to transmit information. S_{in} is partitioned into two equal bins S_0 and S_1 by a value t_{cutoff}. TRCTC transmits a 1-bit by randomly replaying an inter-packet delay from bin S_1 and transmits a 0-bit by randomly replaying an inter-packet delay from bin S_0. Thus, as S_{in} is made up of legitimate traffic, the distribution of TRCTC traffic is approximately equal to the distribution of legitimate traffic.

3.2.1.3 Model-Based Covert Timing Channel

Gianvecchio et al. [45] developed an automated framework for building model-based covert timing channels, which we refer to as MBCTC, to mimic legitimate traffic. MBCTC fits a sample of legitimate traffic to several models, such as Exponential or Weibull, and selects the model with the best fit. MBCTC then uses the inverse distribution function and cumulative distribution function for the selected model as encoding and decoding functions. Based on the inverse transform method for variate generation [71], MBCTC transmits by generating pseudo-random inter-packet delays with hidden information embedded. Thus, as the distribution of the pseudo-random inter-packet delays is determined by the model that approximates legitimate traffic, the distribution of MBCTC is close to that of legitimate traffic. To better model changes in the traffic, MBCTC refits the model in sets of 100 packets.

3.2.1.4 JitterBug

Shah et al. [110] developed a keyboard device called JitterBug that slowly leaks typed information over the network. JitterBug is a passive covert timing channel, so new traffic is not created to transmit information. JitterBug demonstrates how a passive covert timing channel can be positioned so that the target machine does not need to be compromised. A scenario where JitterBug can be used is illustrated in Figure 3.2. In this scenario, an input device is compromised, and the attacker is able to leak typed information over the network. JitterBug operates by creating small delays in keypresses to affect the inter-packet delays of a networked application. JitterBug transmits a 1-bit by increasing an inter-packet delay to a value modulo w milliseconds and transmits a 0-bit by increasing an inter-packet delay to a value modulo $\lceil \frac{w}{2} \rceil$ milliseconds. The timing-window w determines the maximum delay that JitterBug adds to an inter-packet delay. For small values of w, the distribution of JitterBug traffic is very similar to that of the original legitimate traffic. To avoid creating a pattern of inter-packet delays at multiples of w and $\lceil \frac{w}{2} \rceil$, a random sequence s_i is subtracted from the original inter-packet delay before the modulo operation.

3.2.1.5 Other Covert Timing Channels

Berk et al. [13] implemented a simple binary covert timing channel based on the Arimoto-Blahut algorithm, which computes the input distribution that maximizes the channel capacity [?, 14]. Luo et al. [75] designed a combinatorics-based scheme, called Cloak, to transmit information in the ordering of packets within different flows. Cloak can be considered as a storage and timing channel, as the encoding methods require packets and/or flows to be distinguishable by their contents. The same authors also proposed a covert timing channel based on the timing of TCP bursts [76]. Similar to Cloak, El-Atawy et al. [38] built a covert timing channel based on packet ordering and showed how code selection can make this technique effective at evading packet order metrics. Sellke et. al [109] showed that with i.i.d. traffic as cover, it is theoretically possible to create "provably secure" covert timing channels, i.e., covert timing channels that are computationally non-detectable. The same basic proof as [109] can be used to show that TRCTC is computationally non-detectable for i.i.d. cover traffic when its input messages are XOR'd with cryptographically-secure random numbers. Although not a covert timing channel, Giffin et al. [47] showed that low-order bits of the TCP timestamp can be exploited to create a covert storage channel, which is related to timing channels due to the shared statistical properties of timestamps and packet timing.

3.2.1.6 Timing-Based Watermarks

A number of efforts have investigated timing-based watermarking systems [124, 122, 97, 123, 136, 55], which are related to covert timing channels. A timing-based watermarking system is basically a side-channel that is augmented by a low-capacity covert timing channel. Wang et al. [122] proposed a method for watermarking inter-packet delays to track anonymous peer-to-peer voice-over-IP (VoIP) calls. More recently, Houmansadr et. al [55] proposed a subtle watermark called RAINBOW that is non-blind, i.e., it records both incoming and outgoing flows, allowing it to differentiate flows by adding only small delays. By doing so, RAINBOW is able to evade several detection tests, including entropy-based methods. However, the assumptions of timing-based watermarking systems, like RAINBOW, are quite different than those of covert timing channels. The entropy, if any, that is added by a watermarking system can be very small. For example, if a set of flows are naturally differentiable, a watermarking system need not add any delays to differentiate them. Generally, timing-based watermarking systems are passive timing channels in that new traffic is not created. Such systems again demonstrate how a passive timing channel can be positioned so that the target, i.e., the anonymizing network, does not need to be compromised.

3.2.2 Detection Tests

There are two broad classes of detection tests: shape tests and regularity tests. The shape of traffic is described by first-order statistics, e.g., mean, variance, and distribution. The regularity of traffic is described by second or higher-order statistics, e.g., correlations in the data. Note that in previous research the term regularity is sometimes used to refer to frequency-domain regularity [19, 110], whereas here we use this term exclusively to refer to time-domain regularity, i.e., the regularity of a process over time.

3.2.2.1 Kolmogorov-Smirnov Test

Peng et al. [97] showed that the Kolmogorov-Smirnov test is effective to detect watermarked inter-packet delays, a form of timing channel [124]. The watermarked inter-packet delays are shown to have a distribution that is the sum of a normal and a uniform distribution. Thus, the Kolmogorov-Smirnov test can be used to determine if a sample comes from the appropriate distribution. The Kolmogorov-Smirnov test determines whether or not two samples (or a sample and a distribution) differ.

The use of the Kolmogorov-Smirnov test to detect covert timing channels is described in more detail in Section 3.4.1.2. The Kolmogorov-Smirnov test is distribution free, i.e., the test is not dependent on a specific distribution. Thus, the Kolmogorov-Smirnov test is applicable to different types of traffic with different distributions. The Kolmogorov-Smirnov test statistic measures the maximum distance between two empirical distribution functions:

$$KSTEST = \max \mid S_1(x) - S_2(x) \mid, \tag{3.1}$$

where S_1 and S_2 are the empirical distribution functions of the two samples.

3.2.2.2 Regularity Test

Cabuk et al. [19] investigated a method of detecting covert timing channels based on regularity. This detection method, referred to as the regularity test, determines whether or not the variance of the inter-packet delays is relatively constant. This detection test is based on the fact that for most network traffic, the variance of the inter-packet delays changes over time, whereas with covert timing channels, if the encoding scheme does not change over time, then the variance of the inter-packet delays remains relatively constant. The use of the regularity test to detect covert timing channels is discussed in more detail in Section 3.4.1.2. For the regularity test, a sample is separated into sets of w inter-packet delays. Then, for each set, the standard deviation of the set σ_i is computed. The regularity is the standard deviation of the pairwise differences between each σ_i and σ_j for all sets $i < j$.

$$regularity = STDEV\left(\frac{\mid \sigma_i - \sigma_j \mid}{\sigma_i}, i < j, \forall i, j\right) \tag{3.2}$$

3.2.2.3 Other Detection Tests

Cabuk et al. [19] investigated a second method of detecting covert timing channels, referred to as ϵ-similarity, based on measuring the proportion of similar inter-packet delays. The ϵ-similarity test is based on the fact that IPCTC creates clusters of similar inter-packet delays at multiples of the timing-interval. Luo et al. [75] developed a detection method that targets the Cloak channel by measuring the intervals between acknowledgment and data packets. While both detection methods are effective at detecting the specific covert timing channels for which they are designed, namely IPCTC and Cloak, their respective scopes of detection are very limited. In comparison with more generic detection methods, they are less effective at detecting other types of covert timing channels. Berk et al. [13] used a simple mean-max ratio to test for bimodal or multimodal distributions that could be induced by binary or multi-symbol covert timing channels.

3.3 Entropy Measures

In this section, we first describe entropy, conditional entropy, and corrected conditional entropy, and then explain how these measures relate to first-order statistics, second or higher-order statistics, and the regularity or complexity of a process. Finally, we present the design and implementation of the proposed scheme to detect covert timing channels, based on the concept of entropy.

3.3.1 Entropy and Conditional Entropy

The entropy rate, which is the average entropy per random variable, can be used as a measure of complexity or regularity [101, 105]. The entropy rate is defined as the conditional entropy of a sequence of infinite length. The entropy rate is upper-bounded by the entropy of the first-order probability density function or first-order entropy. A simple independent and identically distributed (i.i.d.) process has an entropy rate equal to the first-order entropy. A highly complex process has a high entropy rate, but less than the first-order entropy. A highly regular process has a low entropy rate, zero for a rigid periodic process, i.e., a repeated pattern.

A random process $X = \{X_i\}$ is defined as an indexed sequence of random variables. To give the definition of the entropy rate of a random process, we first define the entropy of a sequence of random variables as:

$$H(X_1, ..., X_m) = -\sum_{X_1,...,X_m} P(x_1, ..., x_m) \log P(x_1, ..., x_m), \tag{3.3}$$

where $P(x_1, ..., x_m)$ is the joint probability $P(X_1 = x_1, ..., X_m = x_m)$.

Then, from the entropy of a sequence of random variables, we define the conditional entropy of a random variable given a previous sequence of random variables as:

$$H(X_m \mid X_1, ..., X_{m-1}) = H(X_1, ..., X_m) - H(X_1, ..., X_{m-1}). \tag{3.4}$$

Lastly, the entropy rate of a random process is defined as:

$$\overline{H}(X) = \lim_{m \to \infty} H(X_m \mid X_1, ..., X_{m-1}). \tag{3.5}$$

The entropy rate is the conditional entropy of a sequence of infinite length and, therefore, cannot be measured for finite samples. Thus, we estimate the entropy rate with the conditional entropy of finite samples. It is also important to note that the definition of entropy rate is for stationary stochastic processes [27] and the extent to which measured data is non-stationary could affect the accuracy of entropy rate estimates.

3.3.2 Corrected Conditional Entropy

The exact entropy rate cannot be measured for finite samples and must be estimated. In practice, we replace probability density functions with empirical probability density functions based on the method of histograms. The data is binned in Q bins. The specific binning strategy being used is important to the overall effectiveness of the test and is discussed in Section 3.3.3. The empirical probability density functions are determined by the proportions of patterns in the data, i.e., the proportion of a pattern is the probability of that pattern. Here a pattern is defined as a sequence of bin numbers. The estimates of the entropy or conditional entropy, based on the empirical probability density functions, are represented as: EN and CE, respectively.

There is a problem with the estimation of $CE(X_m \mid X_1, ..., X_{m-1})$ for some values of m. The conditional entropy tends to zero as m increases, due to limited data. If a specific pattern of length $m-1$ is found only once in the data, then the extension of this pattern to length m will also be found only once. Therefore, the length m pattern can be predicted by the length $m-1$ pattern, and the length m and $m-1$ patterns cancel out. If no pattern of length m is repeated in the data, then $CE(X_m \mid X_{m-1})$ is zero, even for i.i.d. processes.

To solve the problem of limited data, without fixing the length of m, we use the corrected conditional entropy (CCE) [101]. The corrected conditional entropy is defined as:

$$CCE(X_m \mid X_1, ..., X_{m-1}) = CE(X_m \mid X_1, ..., X_{m-1}) + perc(X_m) \cdot EN(X_1), \quad (3.6)$$

where $perc(X_m)$ is the percentage of unique patterns of length m and $EN(X_1)$ is the entropy with m fixed at one, i.e., only the first-order entropy.

The estimate of the entropy rate is the minimum of the corrected conditional entropy over different values of m. The minimum of the corrected conditional entropy is considered to be the best estimate of the entropy rate with the available data. The corrected conditional entropy has a minimum, because the conditional entropy decreases while the corrective term increases. The corrected conditional entropy has been mainly used on biological data, such as electrocardiogram [101] and electroencephalogram data [105]. Although not related to our work, it is interesting to see how such a measure can differentiate the states of complex biological processes. For example, with the electroencephalogram, an increase in the entropy rate indicates a decrease in the depth of anesthesia, i.e., the subject is becoming more conscious.

3.3.3 Binning Strategies

The strategy of binning the data is critical to the overall effectiveness of the test. The binning strategy mainly decides: (1) how the data is partitioned and (2) the bin granularity or the number of bins Q. In previous work, partitioning data into equiprobable bins seems to be most effective [101, 105]. The use of equiprobable bins is illustrated in Figure 3.3, showing the partitioning of Exponential data into bins of equal area. The bins, numbered 1 through 5, are small in width when the proportion of values is high and large in width when the proportion of values is low. Thus, while the bins have different widths, the total area of each bin is equal. The bin number for a value can then be determined based on the cumulative distribution function:

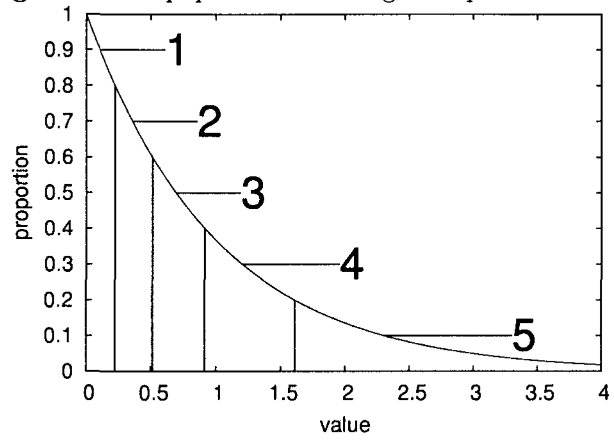
$$bin = \lfloor F(x) * Q \rfloor, \tag{3.7}$$

where F is the cumulative distribution function and x is the value to be binned.

The bin numbers can also be determined based on ranges, e.g., $0.0 < bin_1 \leq 0.22$, $0.22 < bin_2 \leq 0.51$, $0.51 < bin_3 \leq 0.91$, and so on, which requires a search of the ranges to determine the correct bin number for a value. Meanwhile, the cumulative distribution function can determine the correct bin in constant time, which is important for performance when the number of bins is large.

The choice of the number of bins offers a tradeoff. While a larger number of bins retains more information about the distribution of the data, it increases the number

Figure 3.3: Equiprobable Binning of Exponential Data



of possible patterns Q^m and, thus, limits the ability of the test to recognize longer patterns due to the limited data. In contrast, a small number of bins captures less information about the distribution, but is better able to measure the regularity of the data. Therefore, as both strategies have advantages and disadvantages, we use both coarse-grain and fine-grain binning.

To determine the best choice of Q for coarse-grain binning, we run tests on correlated and uncorrelated samples for $Q = 2$ through 10. The correlated samples are 100 traces of 2,000 HTTP inter-packet delays. The uncorrelated samples are random permutations of the correlated samples. We then count the number of uncorrelated samples with scores that overlap with the scores of correlated samples. There is no overlap for the values of $Q = 5$ to 8. Therefore, to retain the ability of the test to recognize longer patterns and measure regularity, we use $Q = 5$ for coarse-grain binning.

It is much simpler to determine the best choice of Q for fine-grain binning. With increasing values of Q, the number of possible patterns Q^m becomes much larger than the size of the sample being tested. At this point, the test scores are dominated by the estimate of the entropy for length one. Then, as we increase the value of Q, the bins continue to become more precise, leading to a better estimate of the entropy for length one than that for smaller values of Q. Therefore, as Q can be made arbitrarily precise, we use $Q = 2^{16} = 65{,}536$ for fine-grain binning.

3.3.4 Implementation Details

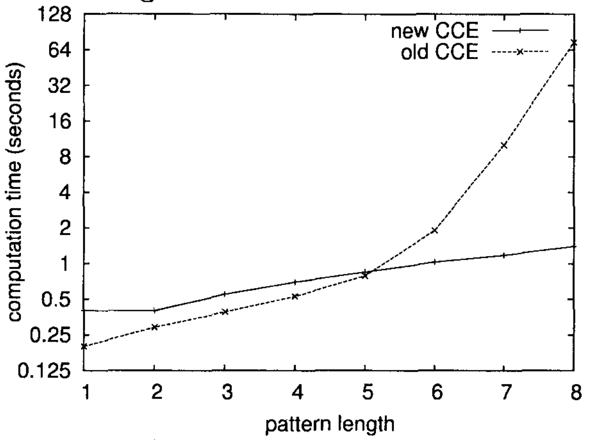
Our design goal is to be effective in detection and efficient in terms of run-time and storage. The efficiency of tests is particularly important if tests are conducted in real-time for online processing of data. Thus, we are careful to optimize our implementation for performance. We implement the corrected conditional entropy in the `C` programming language. The patterns are represented as nodes in a Q-ary tree of height m. The nodes of the tree include pattern counts and links to the nodes with longer patterns. The level of the tree corresponds to the length of patterns. The children of the root are the patterns of length 1. The leaf nodes are the patterns of length m.

To add a new pattern of length m to the tree, we move down the tree towards the leaves, updating the counts of the intermediate nodes and creating new nodes. Thus, when we reach the bottom of the tree, we have counted both the new pattern and all of its sub-patterns. After all patterns of length m are added, we perform a breadth-first traversal. The breadth-first traversal computes the corrected conditional entropy at each level and terminates when the minimum is obtained. If the breadth-first traversal reaches the bottom of the tree without having the minimum, then we must increase m and continue.

The time and space complexities are $O(n \cdot m)$, where n is the size of the sample, if we assume a priori knowledge of the distribution and use the cumulative distribution function to determine the correct bin for each value in constant time. Otherwise, the time complexity increases to $O(n \cdot m \cdot \log(Q))$. In practice, running our program on a sample of size 2,000 with $Q = 5$ and a pattern of length 10 on our test machine, an Intel Pentium D 3.4Ghz, takes 16 milliseconds. However, small changes in the implementation can have significant impact on performance.

To demonstrate this, we evaluate the computation overhead of our implementation and that of a previous implementation [105]. The computation time of both implementations with increasing pattern length is shown in Figure 3.4. For small values of m, our computation time is slightly longer, because of the overhead of creating our data structure. However, as m increases, the previous implementation increases quadratically, whereas our implementation increases linearly. The quadratic growth is caused by the separate processing of patterns of different lengths, i.e., the patterns of length 1, then the patterns of length 2, and so on, which introduces a quadratic term due to the summation of the pattern lengths: $\sum_{i=1}^{m} i = \frac{m^2+m}{2}$.

Figure 3.4: CCE Performance



3.4 Experimental Evaluation

In this section, we validate the effectiveness of our proposed approach through a series of experiments. The focus of these experiments is to determine if our entropy-based methods (entropy and corrected conditional entropy) are able to detect covert timing channels. We test our entropy-based methods against four covert timing channels: IPCTC [19], TRCTC [18], MBCTC [45] and JitterBug [110]. Furthermore, we compare our entropy-based methods to two other detection tests: the Kolmogorov-Smirnov test and the regularity test [19].

The purpose of a detection test is to differentiate covert traffic from legitimate traffic. The performance of a detection test can be measured based on false positive and true positive rates, with low false positive rate and high true positive rate being ideal. In practice, because of the large variation in legitimate network traffic, it is important that tests work well for typical traffic and occasional outliers. If a detection test gives test scores with significant overlap between legitimate and covert samples, then it fails on detection. Therefore, the mean, variance, and distribution of test scores are critical metrics to the performance of a detection test.

3.4.1 Experimental Setup

The defensive perimeter of a network, made up of firewalls and intrusion detection systems, is designed to protect the network from malicious traffic. Typically, only a few specific application protocols, such as HTTP and SMTP, although heavily monitored, are allowed to pass through the defensive perimeter. In addition, other protocols, such as SSH, might be permitted to cross the perimeter but only to specific trusted destinations.

We now consider the scenarios discussed in Section 3.2. In the first scenario, which relates to IPCTC, TRCTC and MBCTC, a compromised machine uses a covert timing channel to communicate with a machine outside the network. For IPCTC, TRCTC and MBCTC, we utilize outgoing HTTP inter-packet delays as the medium, due to the wide acceptance of HTTP for crossing the network perimeter and the high volume of HTTP traffic. In the second scenario, which relates to JitterBug, a compromised input device uses a covert timing channel to leak typed information over the traffic of a networked application. For JitterBug, we utilize outgoing SSH inter-packet delays as the medium, based on the original design [110] and the high volume of keystrokes in interactive network applications.

3.4.1.1 Dataset

The covert and legitimate samples that we use for our experiments are from two datasets: (1) HTTP traces we collected on a medium-size campus network and (2) a dataset obtained from the University of North Carolina at Chapel Hill (UNC). In total, we have 12GB of tcpdump packet header traces (HTTP protocol) that we collected and 79GB of tcpdump packet header traces (all protocols) from the UNC dataset [?]. In our experiments, we use several subsets of the two datasets, including:

- HTTP training set: 200,000 HTTP packets
- HTTP test set: 200,000 HTTP packets
- TRCTC test set: 200,000 HTTP packets
- MBCTC test set: 200,000 HTTP packets
- SSH training set: 200,000 SSH packets
- SSH test set: 200,000 HTTP packets
- JitterBug test set: 200,000 SSH packets

The packets in each dataset are grouped into flows. The flows represent outgoing traffic from a host to a specific port, e.g., port 80 for HTTP or port 22 for SSH. The flows are based on a 3-tuple of source host, destination port, and protocol, rather than a 5-tuple of source address, source port, destination address, destination port, and protocol. The subsets contain 100 samples and each sample has 2,000 packets from a flow.

In our experiments, we test a number of covert samples, which are generated from these subsets and from the encoding methods for IPCTC, TRCTC, MBCTC, and JitterBug. The covert timing channels are configured with the recommended settings from their original works, and we use the most advanced version if multiple versions of a covert timing channel are available. Specifically, IPCTC rotates the timing-interval t amongst 40ms, 60ms and 80ms; TRCTC is the BMC type; and JitterBug subtracts the random sequence s_i before the modulo operation. The input messages transmitted in our tests are random bits generated by a pseudo-random number generator. For TRCTC, we generate the covert samples from a set of 200,000 legitimate HTTP inter-packet delays. For MBCTC, we generate the covert samples from a model that is selected by fitting multiple models to a set of 200,000 legitimate HTTP inter-packet delays. For JitterBug, we generate the covert samples from a set of 200,000 legitimate SSH inter-packet delays. A test machine replays the set of 200,000 SSH inter-packet delays and adds JitterBug delays. Note that our version of JitterBug is implemented in software. A monitoring machine on the campus backbone then collects a trace of the JitterBug traffic, which adds network delays after the addition of JitterBug delays. Since the monitoring machine is only four hops away from the test machine, with a RTT of 0.3ms, the added network delays are small. This JitterBug scenario is illustrated in Figure 3.2, where a defensive perimeter monitors outgoing traffic.

The training sets of legitimate traffic are useful for some of the detection tests. The Kolmogorov-Smirnov test uses the training sets to represent the behavior of legitimate traffic. The Kolmogorov-Smirnov test then measures the distance between the test sample and the training set. The entropy and corrected conditional entropy tests use the training sets to determine the range of each bin, based on equiprobable binning. These tests do not require a priori binning, but doing so improves performance, as the data does not need to be partitioned online.

3.4.1.2 Detection Methodology

In our experiments, we run detection tests on samples of covert and legitimate traffic. We use the resulting test scores to determine if a sample is covert or legitimate as follows. First, we set the targeted false positive rate at 0.01. To achieve this false positive rate, the cutoff scores—the scores that decide whether a sample is legitimate or covert—are set at the 99th or 1st percentile (high scores or low scores for different tests) of legitimate sample scores from the HTTP or SSH training set. Then, samples with scores worse than the cutoff are identified as covert, while samples with scores better than the cutoff are identified as legitimate. The false positive rate is the proportion of legitimate samples in the test set that are wrongly identified as covert, while the true positive rate is the proportion of covert samples in the test set that are correctly identified as covert.

Considering the properties of the detection tests, we can classify them as tests of shape or regularity. The shape of traffic is described by first-order statistics, and the regularity of traffic is described by second or higher-order statistics. The Kolmogorov-Smirnov test and entropy test are tests of shape, while the regularity test and corrected conditional entropy test are tests of regularity. The test scores are interpreted as follows.

In the Kolmogorov-Smirnov test, we measure the distance between the test sample and the training set that represents legitimate behavior. Thus, if the test score is small, it implies that the sample is close to the normal behavior. However, if the sample does not fit the normal behavior well, the test score will be large, indicating the possible occurrence of a covert timing channel. By contrast, in the regularity test, we measure the standard deviation of the normalized standard deviations of sets of 100 packets. If the regularity score is low, then the sample is highly regular, indicating the possible existence of a covert timing channel.

The entropy test estimates the first-order entropy, whereas the corrected conditional entropy test estimates the higher-order entropy. The entropy test is based on the same algorithm as the corrected conditional entropy test, except that the corrective term is not added. The corrected conditional entropy test uses $Q = 5$, whereas the entropy test uses $Q = 65,536$ and m fixed at one. If the entropy test score is low, it suggests a possible covert timing channel, because the sample does not fit the appropriate distribution. If the conditional entropy test score is lower or higher than the cutoff scores, it suggests a possible covert timing channel. When the conditional entropy test score is low, the sample is highly regular. When the conditional entropy test score is high, near the first-order entropy, the sample shows a lack of correlations.

3.4.2 Experimental Results

In the following, we present our experimental results in detail. The four detection tests are: the Kolmogorov-Smirnov test, regularity test, entropy test, and corrected conditional entropy test. The four covert timing channels are: IPCTC, TRCTC, MBCTC, and JitterBug. The experiments are organized by covert timing channels, which are ordered in terms of increasing detection difficulty.

3.4.2.1 IPCTC

Our first set of experiments investigates how the detection tests perform against IPCTC [19]. IPCTC is the simplest among the three covert timing channels being tested and the easiest to detect, because it exhibits abnormality in both shape and regularity. The abnormal shape of IPCTC is caused by the encoding scheme. The encoding scheme encodes a 1-bit by transmitting a packet during an interval, and encodes a 0-bit with no packet transmission. Thus, the number of 0-bits between two 1-bits determines the inter-packet delays. If the bit sequence is random, then we can view the bit sequence as a series of Bernoulli trials and, thus, the inter-packet delays approximate a Geometric distribution. The timing-interval t is rotated among 40 milliseconds, 60 milliseconds, and 80 milliseconds after each 100 packets, as suggested by Cabuk et al. [19], to avoid creating a regular pattern of inter-packet delays at multiples of a single t. However, this instead creates a regular pattern of inter-packet delays at multiples of 20 milliseconds. The regularity of IPCTC is due to the lack of significant correlations between inter-packet delays. That is, the inter-packet delays are determined by the bit sequence being encoded, not by the previous inter-packet delays.

We run each detection test 100 times for 2,000 packet samples of both legitimate traffic and IPCTC traffic. The mean and standard deviation of the test scores are shown in Table 3.1. The detection tests all achieve lower average scores for IPCTC than those for legitimate traffic. The regularity test has a very high standard deviation for legitimate traffic, which suggests that this test is sensitive to variations in the behavior of legitimate traffic. The corrected conditional entropy test has a mean score for covert traffic that appears somewhat close to that of legitimate traffic, 1.96 for legitimate and 2.21 for covert. However, in relative terms, these scores are not that close, since the standard deviation of the corrected conditional entropy test is relatively low. The mean score for IPCTC is much closer to the maximum entropy than to the mean score of legitimate traffic. The maximum entropy is the most uniform possible distribution [27]. The maximum entropy for $Q = 5$ is:

$$H(X) = Q \cdot \frac{1}{Q} \log\left(\frac{1}{Q}\right) = 5 \cdot \frac{1}{5} \log\left(\frac{1}{5}\right) \approx 2.3219 \tag{3.8}$$

The corrected conditional entropy score is bounded from above by the first-order entropy. The first-order entropy is then bounded from above by the maximum entropy. Therefore, the corrected conditional entropy scores for IPCTC are close to the highest values possible.

As shown in Table 3.2, the detection rates for IPCTC (i.e. true positive rates for detecting IPCTC) are 1.0 for all tests except the regularity test, whose detection rate is only 0.54. The regularity test measures sets of 100 packets and the timing-interval t is rotated after each set of 100 packets, so the regularity test observes three distinct variances and accurately measures the regularity of IPCTC. The problem though is not measuring IPCTC, but measuring legitimate traffic. The very high standard deviation of the regularity test against legitimate traffic makes it impossible to differentiate IPCTC from legitimate samples without a higher false positive rate. Moreover, if we increase the timing-interval t to greater than 100 packets, the regularity test observes a different number of packets for each t value within each window, as the sets of t packets overlap with the window at different points, making the test less reliable. However, if we decrease the timing-interval t to much less than 100 packets, the regularity test observes a similar number of packets for each t value within each window and the variance for each window is similar, which makes the test more reliable.

Still, the main problem with the regularity test is its high standard deviation for legitimate traffic. The regularity test is very sensitive to outliers in legitimate traffic. For example, if σ_i is very small, due to a sequence of similar inter-packet delays, and σ_j is average or larger, then $\frac{|\sigma_i - \sigma_j|}{\sigma_i}$ is very large, especially for the values of σ_i close to zero, which are not uncommon. In fact, one such outlier in a sample is more than sufficient to make a covert sample appear to be a legitimate sample. The high variance of the regularity test demonstrates that it is important to examine more than the average test score, since the variance and distribution of test scores are critical to the successful detection of covert timing channels.

Table 3.1: IPCTC Test Scores

	HTTP-TEST		IPCTC	
test	mean	stdev	mean	stdev
$KSTEST$	0.180	0.077	0.708	0.000
$regularity$	35.726	36.635	0.330	0.056
EN	10.454	0.152	6.250	0.028
CCE	1.964	0.149	2.216	0.013

Table 3.2: IPCTC Detection Rates

	HTTP-TEST	IPCTC
test	false positive	true positive
$KSTEST \geq 0.36$	.00	1.00
$regularity \leq 0.41$	.01	.54
$EN \leq 8.56$	.01	1.00
$CCE \geq 2.16$	.01	1.00

3.4.2.2 TRCTC

Our second set of experiments investigates how our detection tests perform against TRCTC [18]. TRCTC is a more advanced covert timing channel that makes use of a replay attack. TRCTC replays a set of legitimate inter-packet delays to approximate the behavior of legitimate traffic. Thus, TRCTC has approximately the same shape as legitimate traffic, but exhibits abnormal regularity, like IPCTC. The regularity of TRCTC, like IPCTC, is due to the lack of significant correlations between inter-packet delays. Although TRCTC replays inter-packet delays, the replay is in random order, as determined by the bit sequence that is being encoded, thus breaking the correlations in the original inter-packet delays.

We run each detection test 100 times for 2,000 packet samples of both legitimate traffic and TRCTC traffic. The mean and standard deviation of the test scores are shown in Table 3.3. The test scores for TRCTC and legitimate traffic are approximately equal for the Kolmogorov-Smirnov and entropy tests. These tests strictly measure first-order statistics, and, as such, are not able to detect TRCTC. The regularity test achieves a much lower average score for TRCTC than that for legitimate traffic, which is due to the similar variance between groups of packets in TRCTC. However, the standard deviation of the regularity test is again very high for legitimate traffic and, this time, is high for covert traffic as well. At the same time, the corrected conditional entropy test gives similar results to those for IPCTC. The corrected conditional entropy test has a mean score for TRCTC that appears somewhat close to that of legitimate, 1.96 for legitimate and 2.21 for covert. However, if we examine the distribution of test scores for TRCTC and legitimate traffic, as illustrated in Figure 3.5, then we can see that, although some scores are in adjacent bins, there is no overlap between the distributions. Furthermore, the distribution of legitimate test scores is strongly skewed to the left, away from the distribution of TRCTC test scores. The detection rates for TRCTC, as shown in Table 3.4, are very low (0.04 or less) for all the detection tests except the corrected conditional entropy test, which has a detection rate of 1.0. The corrected conditional entropy test scores of TRCTC are again close to the maximum entropy, therefore the corrected conditional entropy test is successful in detecting TRCTC.

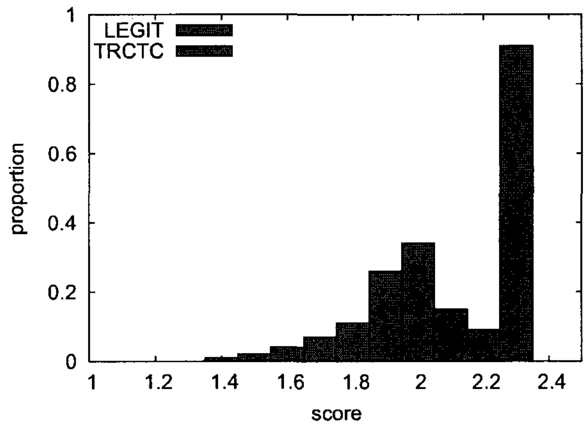
Table 3.3: TRCTC Test Scores

	HTTP-TEST		TRCTC	
test	mean	stdev	mean	stdev
KSTEST	0.180	0.077	0.180	0.077
regularity	35.726	36.635	7.845	9.324
EN	10.454	0.152	10.454	0.152
CCE	1.964	0.149	2.217	0.012

Table 3.4: TRCTC Detection Rates

	HTTP-TEST	TRCTC
test	false positive	true positive
$KSTEST \geq 0.36$	.00	.01
$regularity \leq 0.41$	.01	.04
$EN \leq 8.56$	.01	.02
$CCE \geq 2.16$	.01	1.00

Figure 3.5: Distribution of CCE Test Scores for TRCTC



3.4.2.3 MBCTC

Our third set of experiments investigates how our detection tests perform against MBCTC [45]. MBCTC is a more advanced covert timing channel that exploits traffic modeling to mimic legitimate traffic. The traffic model is determined by using maximum likelihood estimation (MLE) to determine model parameters and then selecting the model with the lowest root mean squared error (RMSE) from several models. The model selected for legitimate HTTP traffic is Weibull with a mean scale parameter λ of 0.125 and a mean shape parameter k of 0.426. With these parameters, the mean inter-packet delay is 0.3524, approximately 3 packets per second. The model is then refitted in sets of 100 packets to better model changes in the traffic over time. Thus, MBCTC has a similar shape to legitimate traffic, due to modeling the distribution, and a similar regularity for sets of 100 packets or more, due to the refitting process.

We run each detection test 100 times for 2,000 packet samples of both legitimate traffic and MBCTC traffic. The mean and standard deviation of the test scores are shown in Table 3.5. The test scores of MBCTC are higher than those of legitimate traffic for the Kolmogorov-Smirnov test, though less than the standard deviation, due to the model being very close but not a perfect fit. The regularity test achieves a lower average score for MBCTC than that of legitimate traffic, though the standard deviation is again very high for legitimate traffic and covert traffic. The entropy test scores of MBCTC are higher on average than those of legitimate traffic, indicating that MBCTC traffic is consistently a somewhat close fit to the legitimate traffic distribution. The corrected conditional entropy test scores are significantly lower for MBCTC than for legitimate traffic. However, when we examine the distribution of test scores for MBCTC and legitimate traffic, as illustrated in Figure 3.6, we can see that there is a slight overlap between the distributions. This shows that the refitting process used by MBCTC, i.e., changing the model after each set of 100 packets, is relatively successful, but not sufficient to capture the true regularity of legitimate traffic. In particular, MBCTC traffic is more regular over time than legitimate traffic, i.e., the sequences of inter-packet delays are more predictable. For example, if a burst occurs, then the expected value of the model will be small and MBCTC will generate a larger portion of small inter-packet delays for the next 100 inter-packet delays. As a result, small inter-packet delays will be more likely to be followed by small inter-packet delays in MBCTC traffic than in legitimate traffic, which results in lower scores for the corrected conditional entropy test. The detection rates of MBCTC, as shown in Table 3.4, are very low (0.04 or less) for all the detection tests except the entropy test and the corrected conditional entropy test. The entropy test is able to sometimes detect MBCTC, with a detection rate of 0.55. The corrected conditional entropy test is very successful in detecting MBCTC, with a detection rate of 0.95.

Table 3.5: MBCTC Test Scores

	HTTP-TEST		MBCTC	
test	mean	stdev	mean	stdev
$KSTEST$	0.180	0.077	0.208	0.073
$regularity$	35.726	36.635	18.440	22.605
EN	10.454	0.152	10.739	0.078
CCE	1.964	0.149	1.156	0.223

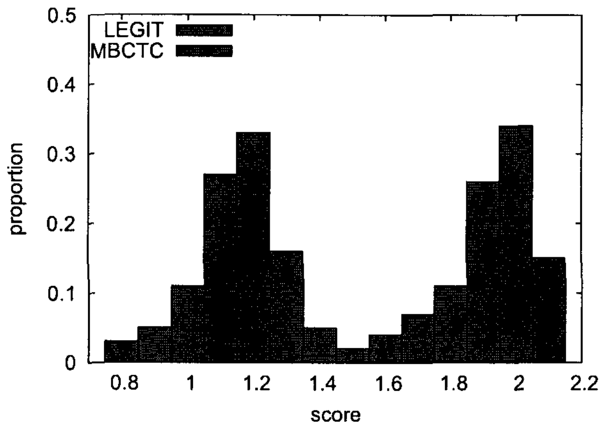
Table 3.6: MBCTC Detection Rates

	HTTP-TEST	MBCTC
test	false positive	true positive
$KSTEST \geq 0.36$	.00	.03
$regularity \leq 0.41$	.01	.02
$EN \geq 10.74$	.01	.55
$CCE \leq 1.50$	.00	.95

3.4.2.4 JitterBug

Our fourth set of experiments investigates how our detection tests perform against JitterBug [110]. JitterBug is a passive covert timing channel, so no additional traffic is generated to transmit information. Instead, JitterBug manipulates the inter-packet delays of existing legitimate traffic. The timing-window w, which determines the maximum delay that JitterBug adds, is set at 20 milliseconds, as suggested by Shah

Figure 3.6: Distribution of CCE Test Scores for MBCTC



et al. [110]. The average inter-packet delay of the original SSH traffic is 1.264 seconds, whereas, with JitterBug, the average inter-packet delay is 1.274 seconds. In addition, while 10 milliseconds on average might be noticeable with other protocols, SSH traffic has a small proportion of short inter-packet delays, i.e., only about 20% of inter-packet delays are less than 30ms in the training set. Therefore, because of having legitimate traffic as a base and only slightly increasing the inter-packet delays, JitterBug is able to retain much of the original correlation from the legitimate traffic. Moreover, by slightly increasing the inter-packet delays, JitterBug only slightly affects the original shape. Thus, JitterBug has similar shape and regularity to legitimate traffic.

Also JitterBug is very difficult to detect for several other reasons. From a practical perspective, the machine itself has not been compromised, so conventional host-based intrusion detection methods fail. Moreover, the traffic is encrypted, so the contents of the packets cannot be used to predict the appropriate behavior. Additionally, the position of JitterBug, between the machine and the human, further complicates detection because of the variation in human behavior, i.e., different typing characteristics. However, as JitterBug is a covert timing channel and transmits information, there is some affect on the entropy of the original process.

We run each detection test 100 times for 2,000 packet samples of both legitimate traffic and JitterBug traffic. The mean and standard deviation of the test scores are shown in Table 3.7. The test scores for JitterBug and legitimate traffic are close to each other for all the tests except the entropy test. If we examine the distribution of entropy test scores for JitterBug and legitimate traffics, as illustrated in Figure 3.5, we can see that the distributions of JitterBug and legitimate test scores are quite distinct. The detection rates for JitterBug shown in Table 3.8, are very low (0.04 or less) for all the detection tests except the entropy test, which has a detection rate of 1.0. Note that the other tests do detect some difference between JitterBug and legitimate traffic, but the differences are so small that it is impossible for these tests to differentiate JitterBug from legitimate traffic without a much higher false positive rate. Although the corrected conditional entropy test is successful at detecting all the other covert timing channels, it is unable to detect JitterBug. The corrected conditional entropy test bins the data into $Q = 5$ bins. For SSH traffic, the typical bin ranges (based on equiprobable binning) are $0.0 < bin_1 \leq 0.032$, $0.032 < bin_2 \leq 0.088$, $0.088 < bin_3 \leq 0.160$, $0.160 < bin_4 \leq 0.305$, and $0.305 < bin_5$. JitterBug adds a maximum of 20ms (10ms on average) to the inter-packet delays, so the bin numbers for inter-packet delays are rarely changed. Therefore, the corrected conditional entropy scores of JitterBug traffic are close to those of the original legitimate SSH traffic. In short, the corrected conditional entropy test is simply insensitive to small changes in the distribution.

In contrast, the entropy test is able to detect JitterBug. The entropy test uses a large number of bins, with bin widths determined by the distribution of legitimate traffic. The entropy test measures how uniformly the inter-packet delays are distributed into the bins, and how uniformly the inter-packet delays fit the legitimate traffic distribution. JitterBug creates small changes throughout the distribution. Since these changes fall within the variance that is typical of legitimate traffic, the tests that measure the maximum distance, like the Kolmogorov-Smirnov test, fail to detect the changes. However, the entropy test is sensitive to such changes throughout the distribution. JitterBug increases the inter-packet delays and, due to the rotating window, redistributes the inter-packet delays in an Equilikely distribution. However, the increases do not follow the legitimate distribution, leading to slight increases or decreases in the proportion of inter-packet delays for different bins. The entropy test measures how evenly the inter-packet delays are distributed into the bins, with the legitimate traffic distribution resulting in the most even or uniform distribution of bins and the most entropy, since the bins are sized to be equiprobable for the legitimate distribution. Therefore, the entropy test score for JitterBug is lower than that for legitimate traffic, which can be easily detected.

Table 3.7: JitterBug Test Scores

	SSH-TEST		JitterBug	
test	mean	stdev	mean	stdev
$KSTEST$	.270	.133	.273	.123
$regularity$	6.230	5.847	6.038	5.624
EN	10.663	0.374	8.199	0.720
CCE	1.779	0.261	1.837	0.220

Table 3.8: JitterBug Detection Rates

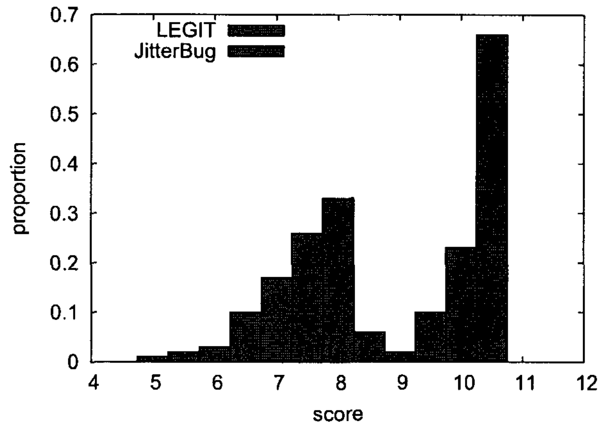
	SSH-TEST	JitterBug
test	false positive	true positive
$KSTEST \geq 0.60$	.02	.03
$regularity \leq 0.15$	.03	.03
$EN \leq 8.84$	.01	1.00
$CCE \geq 2.16$	.01	.04

3.4.2.5 All Channels - Variable Sample Size

Our last set of experiments investigates how our detection tests perform with different sample sizes against all four covert timing channels, IPCTC, TRCTC, MBCTC, and JitterBug. We vary sample sizes from 500 to 2,000 inter-packet delays for the entropy test and the corrected conditional entropy test. The sample size is important because it determines the amount of time it takes to detect a covert timing channel, and thus, the amount of information that a covert timing channel can transmit before it is detected. Of course, the faster a covert timing channel can be detected, the less information it can transmit. However, there is a tradeoff between detection speed and detection accuracy. While a smaller sample size means faster detection, it tends to be less accurate compared to larger sample sizes.

The true positive rates for the entropy test against IPCTC, TRCTC, MBCTC, and JitterBug with 500 to 2,000 inter-packet delays are shown in Figure 3.8. The

Figure 3.7: Distribution of EN Test Scores for JitterBug



true positive rates degrade at different rates in different covert timing channels. On one extreme, for IPCTC there is no decrease in true positive rate and it is easily detected with as little as 500 inter-packet delays. The pattern of IPCTC is obvious, so there is no need for a large amount of data. On the other extreme, the true positive rates of JitterBug degrade more rapidly with the decrease of sample size, and it is difficult to detect JitterBug with less than 1,600 inter-packet delays. JitterBug is more subtle. It adds only small delays and has a relatively low capacity, so its inter-packet delays are indistinguishable from normal without more traffic. In the middle, the true positive rates of MBCTC degrade gradually with the decrease of sample size, starting at 0.55 and ending at 0.14, showing approximately a linear relationship between its true positive rate and its sample size. Lastly, TRCTC is not detected by the entropy test, so its true positive rates remain close to zero.

The true positive rates for the corrected conditional entropy test against IPCTC, TRCTC, MBCTC, and Jitterbug with 500 to 2,000 inter-packet delays are shown in

Figure 3.9. IPCTC and TRCTC demonstrate a similar trend in their true positive rates. Both have true positive rates close to 1.0 with more than 700 inter-packet delays and then degrade quickly with the decrease of sample size. As neither covert timing channel attempts to capture inter-dependencies between inter-packet delays, this likely indicates that the minimum sample size required by the corrected conditional entropy test for accurate detection is around 700. The true positive rates of MBCTC again decline gradually with the decrease of sample size, starting at 0.95 and ending at 0.27, similar to the corresponding entropy test results. JitterBug is not detected by the corrected conditional entropy test, so its true positive rates are close to zero for all sample sizes.

Overall, combining the results of both tests, we can see that IPCTC and TRCTC are easier to be detected than MBCTC and Jitterbug when sample size is small. IPCTC and TRCTC can be accurately detected at the true positive rates of 1.0, with as little as 500 inter-packet delays and 1,000 inter-packet delays, respectively. MBCTC and JitterBug are much more difficult to detect, and they require close to 2,000 inter-packet delays or more for accurate detection. These results are attributed to the fact that MBCTC and JitterBug effectively capture both traffic shape and traffic regularity, while TRCTC only captures traffic shape and IPCTC captures neither of these two properties.

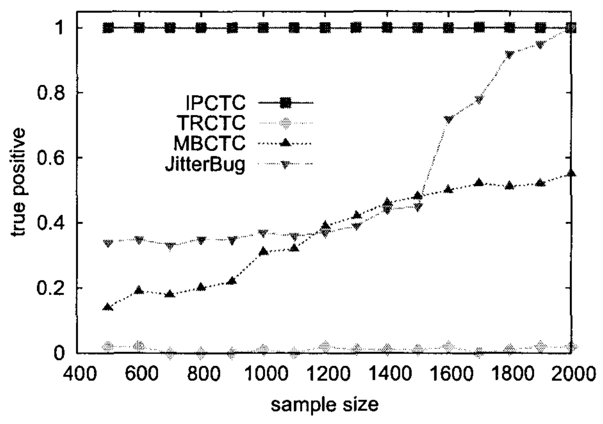
3.4.3 Discussion

The detection tests that we present are all able to detect some covert timing channels under certain conditions. However, the previous methods fail for detecting most of the tested covert timing channels. One major reason lies in the high variation of legitimate traffic. For example, the regularity test exhibits obvious weakness in this regard. Interestingly, the regularity test is the only test, other than the corrected conditional entropy test, that achieves lower average scores for all the covert timing channels. However, due to the high standard deviation of the regularity test in measuring legitimate traffic, the regularity test is not an effective detection method.

The other main reason lies in the properties of covert traffic. For example, while the Kolmogorov-Smirnov test is better able to deal with legitimate traffic variation, it has problems with covert timing channels whose distribution is very close to that of legitimate traffic. The Kolmogorov-Smirnov test measures the maximum distance between the two distributions, rather than measuring differences throughout the distribution. Thus, when the distribution of covert traffic is very close to that of legitimate traffic, the variance of the test scores is sufficiently large so that the test cannot differentiate covert traffic from legitimate traffic.

Our entropy-based approach proves more effective than previous schemes. Based on the advantages of different binning strategies, we make use of both entropy and corrected conditional entropy for detecting covert timing channels. The entropy test is sensitive to small changes throughout the distribution. However, for a covert timing channel whose distribution is nearly identical to that of legitimate traffic, the entropy test fails. By contrast, the corrected conditional entropy test measures the regularity or complexity of the traffic, rather than the distribution. Thus, it is effective to detect such a covert timing channel. However, if the original correlations of traffic are retained and the distribution is changed, then the conditional entropy test fails; but the entropy test works in this scenario by detecting slight changes in the distribution. Therefore, when both tests are combined, our entropy-based approach is effective in detecting all the tested covert timing channels.

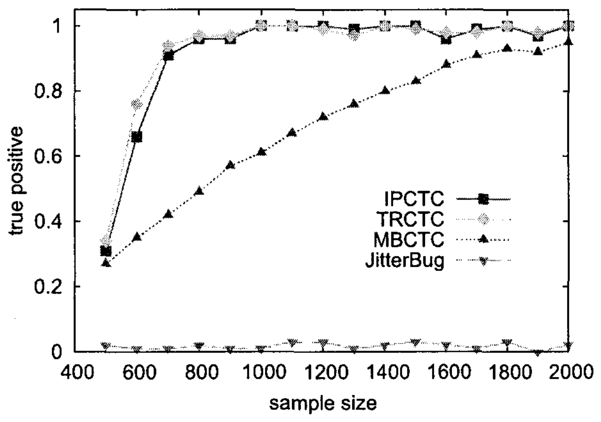
Figure 3.8: EN True Positive Rate vs. Sample Size



3.5 Potential Countermeasures

In this section, we discuss possible countermeasures that could be used to harden covert timing channels against our entropy-based approach. Our discussion focuses on TRCTC, MBCTC and JitterBug. TRCTC and MBCTC are detected by the corrected conditional entropy test and JitterBug is detected by the entropy test.

Figure 3.9: CCE True Positive Rate vs. Sample Size



In an attempt to evade the corrected conditional entropy test, TRCTC could be redesigned to replay longer correlated sequences of inter-packet delays. The corrected conditional entropy test could counter this technique for short sequences by increasing the minimum pattern length. Of course, with increasing sequence length, the corrected conditional entropy test would lose its capability to measure regularity, because of the issues discussed in Section 3.3, unless the sample size were increased. However, this is not a significant threat, because replaying long correlated sequences of inter-packet delays would greatly reduce the capacity of TRCTC. In an attempt to evade the corrected conditional entropy test, MBCTC could be changed to refit the model more frequently so as to better capture the regularity of traffic. Moreover, MBCTC could be redesigned to model conditional distributions to better capture inter-dependencies in traffic.

In an attempt to evade the entropy test, JitterBug could be reconfigured to use a smaller timing-window w. Eventually, as w becomes smaller, the entropy test would need a larger sample size to detect the JitterBug. However, using a smaller timing-window would, similar to our discussion of TRCTC, reduce the capacity of JitterBug. Additionally, JitterBug could be changed to transmit packets at more precise timing than milliseconds, as the millisecond-level precision could create a detectable pattern when the network delays are small. As another alternative, since a large number of inter-packet delays are required to detect JitterBug, JitterBug could attempt to transmit with fewer inter-packet delays than the minimum required for the entropy test. However, there is a problem with this approach. JitterBug uses forward error correction with repeated transmissions. This mechanism provides reliable communication even if packets are lost or some of the perturbed keystrokes go to a non-network application, neither of which can be detected by a JitterBug embedded in the keyboard. By reducing the number of repetitions, JitterBug could evade detection, but could also fail to deliver its message. It remains an open question whether these countermeasures would be practical.

3.6 Conclusion and Future Work

In this chapter, we introduced an entropy-based technique to detect covert-timing channels by employing both entropy and corrected conditional entropy. We designed and implemented the proposed entropy-based detection tool. The development of this tool addresses a number of non-trivial design issues, including efficient use of data structures, data partition, bin granularity, and pattern length. We observed that as bin granularity increases, entropy estimates become more precise, whereas corrected conditional entropy estimates become less precise. Therefore, based on this observation, we utilized the fine-binned entropy estimation and the coarse-binned corrected conditional entropy estimation for covert timing channel detection.

We then applied our entropy-based techniques for detecting covert timing channels. The corrected conditional entropy test is able to detect the covert timing channels with abnormal regularity, while the entropy test is able to detect the covert timing channels with abnormal shape. Our experimental results show that the combination of entropy and corrected conditional entropy is capable of detecting a variety of covert timing channels. In contrast, for a covert timing channel whose distribution is close to that of legitimate traffic, all the previous detection methods fail.

There are a number of possible directions for our future work. We plan to further investigate the possible countermeasures that could be used by attackers to evade entropy-based detection. We also plan to explore the connection between our entropy-based detection methods and the entropy that relates to covert timing channel capacity. We believe that the exploration could lead to better detection methods or lower overall bounds on the capacity of covert timing channels.

Chapter 4

Measurement and Classification of Chat Bots

Internet chat is a popular application that enables real-time text-based communication. Millions of people around the world use Internet chat to exchange messages and discuss a broad range of topics on-line. Internet chat is also a unique networked application, because of its human-to-human interaction and low bandwidth consumption [32]. However, the large user base and open nature of Internet chat make it an ideal target for malicious exploitation.

The abuse of chat services by automated programs, known as *chat bots*, poses a serious threat to on-line users. Chat bots have been found on a number of chat systems, including commercial chat networks, such as AOL [99, 56], Yahoo! [98, 68, 112, 85, 84] and MSN [57], and open chat networks, such as IRC and Jabber. There are also reports of bots in some non-chat systems with chat features, including online games, such as World of Warcraft [28, 107] and Second Life [93]. Chat bots exploit these on-line systems to send spam, spread malware, and mount phishing attacks.

So far, the efforts to combat chat bots have focused on two different approaches: (1) keyword-based filtering and (2) human interactive proofs. The keyword-based message filters, used by third party chat clients [131, 134], suffer from high false negative rates because bot makers frequently update chat bots to evade published keyword lists. The use of human interactive proofs, such as CAPTCHAs [3], is also ineffective because bot operators assist chat bots in passing the tests to log into chat rooms [85, 84]. In August 2007, Yahoo! implemented CAPTCHA to block bots from entering chat rooms, but bots are still able to enter chat rooms in large numbers. There are online petitions against both AOL and Yahoo! [99, 98], requesting that the chat service providers address the growing bot problem. While on-line systems are besieged with chat bots, no systematic investigation on chat bots has been conducted. The effective detection system against chat bots is in great demand but still missing.

In the chapter, we first perform a series of measurements on a large commercial chat network, Yahoo! chat, to study the behaviors of chat bots and humans in on-line chat systems. Our measurements capture a total of 14 different types of chat bots. The different types of chat bots use different triggering mechanisms and text obfuscation techniques. The former determines message timing, and the latter determines message content. Our measurements also reveal that human behavior is more complex than bot behavior, which motivates the use of entropy rate, a measure of complexity, for chat bot classification. Based on the measurement study, we propose a classification system to accurately distinguish chat bots from humans. There are two main components in our classification system: (1) an entropy classifier and (2) a machine-learning classifier. Based on the characteristics of message time and size, the entropy classifier measures the complexity of chat flows and then classifies them as bots or humans. In contrast, the machine-learning classifier is mainly based on message content for detection. The two classifiers complement each other in chat bot detection. While the entropy classifier requires more messages for detection and, thus, is slower, it is more accurate to detect unknown chat bots. Moreover, the entropy classifier helps train the machine-learning classifier. The machine learning classifier requires less messages for detection and, thus, is faster, but cannot detect most unknown bots. By combining the entropy classifier and the machine-learning classifier, the proposed classification system is highly effective to capture chat bots, in terms of accuracy and speed. We conduct experimental tests on the classification system, and the results validate its efficacy on chat bot detection.

4.1 Background and Related Work

4.1.1 Chat Systems

Internet chat is a real-time communication tool that allows on-line users to communicate via text in virtual spaces, called chat rooms or channels. There are a number of protocols that support chat [59], including IRC, Jabber/XMPP, MSN/WLM (Microsoft), OSCAR (AOL), and YCHT/YMSG (Yahoo!). The users connect to a chat server via chat clients that support a certain chat protocol, and they may browse and join many chat rooms featuring a variety of topics. The chat server relays chat messages to and from on-line users. A chat service with a large user base might employ multiple chat servers. In addition, there are several multi-protocol chat clients, such as Pidgin (formerly GAIM) and Trillian, that allow a user to join different chat systems.

Although IRC has existed for a long time, it has not gained mainstream popularity. This is mainly because its console-like interface and command-line-based operation are not user-friendly. The recent chat systems improve user experience by using graphic-based interfaces, as well as adding attractive features such as avatars, emoticons, and audio-video communication capabilities. Our study is carried out on the Yahoo! chat network, one of the largest and most popular commercial chat systems.

Yahoo! chat uses proprietary protocols, in which the chat messages are transmitted in plain-text, while commands, status and other meta data are transmitted as encoded binary data. Unlike those on most IRC networks, users on the Yahoo! chat network cannot create chat rooms with customized topics because this feature is disabled by Yahoo! to prevent abuses [82]. In addition, users on Yahoo! chat are required to pass a CAPTCHA word verification test in order to join a chat room. This recently-added feature is to guard against a major source of abuse—bots.

4.1.2 Chat Bots

The term *bot*, short for robot, refers to automated programs, that is, programs that do not require a human operator. A chat bot is a program that interacts with a chat service to automate tasks for a human, e.g., creating chat logs. The first-generation chat bots were designed to help operate chat rooms, or to entertain chat users, e.g., quiz or quote bots. However, with the commercialization of the Internet, the main enterprise of chat bots is now sending chat spam. Chat bots deliver spam URLs via either links in chat messages or user profile links. A single bot operator, controlling a few hundred chat bots, can distribute spam links to thousands of users in different chat rooms, making chat bots very profitable to the bot operator who is paid per-click through affiliate programs. Other potential abuses of bots include spreading malware, phishing, booting, and similar malicious activities.

A few countermeasures have been used to defend against the abuse of chat bots, though none of them are very effective. On the server side, CAPTCHA tests are used by Yahoo! chat in an effort to prevent chat bots joining chat rooms. However, this defense becomes ineffective as chat bots bypass CAPTCHA tests with human assistance. We have observed that bots continue to join chat rooms and sometimes even become the majority members of a chat room after the deployment of CAPTCHA tests. Third-party chat clients filter out chat bots, mainly based on key words or key phrases that are known to be used by chat bots. The drawback with this approach is that it cannot capture those unknown or evasive chat bots that do not use the known key words or phrases.

4.1.3 Related Work

Dewes et al. [32] conducted a systematic measurement study of IRC and Web-chat traffic, revealing several statistical properties of chat traffic. (1) Chat sessions tend to last for a long time, and a significant number of IRC sessions last much longer than Web-chat sessions. (2) Chat session inter-arrival time follows an exponential distribution, while the distribution of message inter-arrival time is not exponential. (3) In terms of message size, all chat sessions are dominated by a large number of small packets. (4) Over an entire session, typically a user receives about 10 times as much data as he sends. However, very active users in Web-chat and automated scripts used in IRC may send more data than they receive.

There is considerable overlap between chat and instant messaging (IM) systems, in terms of protocol and user base. Many widely used chat systems such as IRC predate the rise of IM systems, and have great impact upon the IM system and protocol design. In return, some new features that make the IM systems more user-friendly have been back-ported to the chat systems. For example, IRC, a classic chat system, implements a number of IM-like features, such as presence and file transfers, in its current versions. Some messaging service providers, such as Yahoo!, offer both chat and IM accesses to their end-user clients. With this in mind, we outline some related work on IM systems. Liu et al. [73] explored client-side and server-side methods for detecting and filtering IM spam or *spim*. However, their evaluation is based on a corpus of short e-mail spam messages, due to the lack of data on spim. In [77], Mannan et al. studied IM worms, automated malware that spreads on IM systems using the IM contact list. Leveraging the spreading characteristics of IM malware, Xie et al. [130] presented an IM malware detection and suppression system based on the honeypot concept.

Botnets consist of a large number of slave computing assets, which are also called "bots". However, the usage and behavior of bots in botnets are quite different from those of chat bots. The bots in botnets are malicious programs designed specifically to run on compromised hosts on the Internet, and they are used as platforms to launch a variety of illicit and criminal activities such as credential theft, phishing, distributed denial-of-service attacks, etc. In contrast, chat bots are automated programs designed mainly to interact with chat users by sending spam messages and URLs in chat rooms. Although having been used by botnets as command and control mechanisms [49, 4], IRC and other chat systems do not play an irreplaceable role in botnets. In fact, due to the increasing focus on detecting and thwarting IRC-based botnets [30, 52, 53], recently emerged botnets, such as Phatbot, Nugache, Slapper, and Sinit, show a tendency towards using P2P-based control architectures [123].

Chat spam shares some similarities with email spam. Like email spam, chat spam contains advertisements of illegal services and counterfeit goods, and solicits human users to click spam URLs. Chat bots employ many text obfuscation techniques used by email spam such as word padding and synonym substitution. Since the detection of email spam can be easily converted into the problem of text classification, many content-based filters utilize machine-learning algorithms for filtering email spam. Among them, Bayesian-based statistical approaches [51, 135, 17, 137, 72] have achieved high accuracy and performance. Although very successful, Bayesian-based spam detection techniques still can be evaded by carefully crafted messages [128, 74, 64].

4.2 Measurement

In this section, we detail our measurements on Yahoo! chat, one of the most popular commercial chat services. The focus of our measurements is on public messages posted to Yahoo! chat rooms. The logging of chat messages is available on the standard Yahoo! chat client, as well as most third party chat clients. Upon entering chat, all chat users are shown a disclaimer from Yahoo! that other users can log their messages. However, we consider the contents of the chat logs to be sensitive, so we only present fully-anonymized statistics.

Our data was collected between August and November of 2007. In late August, Yahoo! implemented a CAPTCHA check on entering chat rooms [85, 7], creating technical problems that made their chat rooms unstable for about two weeks [5, 6]. At the same time, Yahoo! implemented a protocol update, preventing most third party chat clients, used by a large proportion of Yahoo! chat users, from accessing the chat rooms. In short, these upgrades made the chat rooms difficult to be accessed for both chat bots and humans. In mid to late September, both chat bot and third party client developers updated their programs. By early October, chat bots were found in Yahoo! chat [84], possibly bypassing the CAPTCHA check with human assistance. Due to these problems and the lack of chat bots in September and early October, we perform our analysis on August and November chat logs. In August and November, we collected a total of 1,440 hours of chat logs. There are 147 individual chat logs from 21 different chat rooms. The process of reading and labeling these chat logs required about 100 hours. To the best of our knowledge, we are the first in the large scale measurement and classification of chat bots.

4.2.1 Log-Based Classification

In order to characterize the behavior of human users and that of chat bots, we need two sets of chat logs pre-labeled as bots and humans. To create such datasets, we perform log-based classification by reading and labeling a large number of chat logs. The chat users are labeled in three categories: human, bot, and ambiguous.

The log-based classification process is a variation of the Turing test. In a standard Turing test [118], the examiner converses with a test subject (a possible machine) for five minutes, and then decides if the subject is a human or a machine. In our classification process, the examiner observes a long conversation between a test subject (a possible chat bot) and one or more third parties, and then decides if the subject is a human or a chat bot. In addition, our examiner checks the content of URLs and typically observes multiple instances of the same chat bot, which further improve our classification accuracy. Moreover, given that the best practice of current artificial intelligences [116] can rarely pass a non-restricted Turing test, our classification of chat bots should be very accurate.

Although a Turing test is subjective, we outline a few important criteria. The main criterion for being labeled as human is a high proportion of specific, intelligent, and human-like responses to other users. In general, if a user's responses suggest more advanced intelligence than current state-of-the-art AI [116], then the user can be labeled as human. The ambiguous label is reserved for non-English, incoherent, or non-communicative users. The criteria for being classified as bot are as follows. The first is the lack of the intelligent responses required for the human label. The second is the repetition of similar phrases either over time or from other users (other instances of the same chat bot). The third is the presence of spam or malware URLs in messages or in the user's profile.

4.2.2 Analysis

In total, our measurements capture 14 different types of chat bots. The different types of chat bots are determined by their triggering mechanisms and text obfuscation schemes. The former relates to message timing, and the latter relates to message content. The two main types of triggering mechanisms observed in our measurements are timer-based and response-based. A timer-based bot sends messages based on a timer, which can be periodic (i.e., fixed time intervals) or random (i.e., variable time intervals). A response-based bot sends messages based on programmed responses to specific content in messages posted by other users.

There are many different kinds of text obfuscation schemes. The purpose of text obfuscation is to vary the content of messages and make bots more difficult to recognize or appear more human-like. We observed four basic text obfuscation methods that chat bots use to evade filtering or detection. First, chat bots introduce random characters or space into their messages, similar to some spam e-mails. Second, chat bots use various synonym phrases to avoid obvious keywords. By this method, a template with several synonyms for multiple words can lead to thousands of possible messages. Third, chat bots use short messages or break up long messages into multiple messages to evade message filters that work on a message-by-message basis. Fourth, and most interestingly, chat bots replay human phrases entered by other chat users.

According to our observation, the main activity of chat bots is to send spam links to chat users. There are two approaches that chat bots use to distribute spam links in chat rooms. The first is to post a message with a spam link directly in the chat room. The second is to enter the spam URL in the chat bot's user profile and then convince the users to view the profile and click the link. Our logs also include some examples of malware spreading via chat rooms. The behavior of malware-spreading chat bots is very similar to that of spam-sending chat bots, as both attempt to lure human users to click links. Although we did not perform detailed malware analysis on links posted in the chat rooms and Yahoo! applies filters to block links to known malicious files, we found several worm instances in our data. There are 12 W32.Imaut.AS [114] worms appeared in the August chat logs, and 23 W32.Imaut.AS worms appeared in the November chat logs. The November worms attempted to send malicious links but were blocked by Yahoo! (the malicious links in their messages being removed), however, the August worms were able to send out malicious links.

The focus of our measurements is mainly on short term statistics, as these statistics are most likely to be useful in chat bot classification. The two key measurement metrics in this study are inter-message delay and message size. Based on these two metrics, we profile the behavior of human and that of chat bots. Among chat bots, we further divide them into four different groups: periodic bots, random bots, responder bots, and replay bots. With respect to these short-term statistics, human and chat bots behave differently, as shown below.

4.2.2.1 Humans

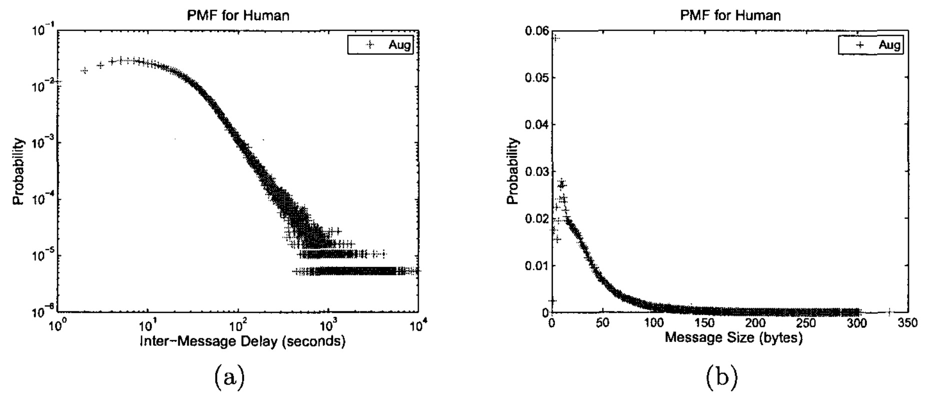


Figure 4.1: Distribution of Human Inter-Message Delay (a) and Message Size (b)

Figure 4.1 shows the probability distributions of human inter-message delay and message size. Since the behavior of humans is persistent, we only draw the probability mass function (pmf) curves based on the August data. The previous study on Internet chat systems [32] observed that the distribution of inter-message delay in chat systems was heavy tailed. In general our measurement result conforms to that observation. The body part of the pmf curve in Figure 4.1 (a) (log-log scale) can be linearly fitted, indicating that the distribution of human inter-message delays follows a power law.

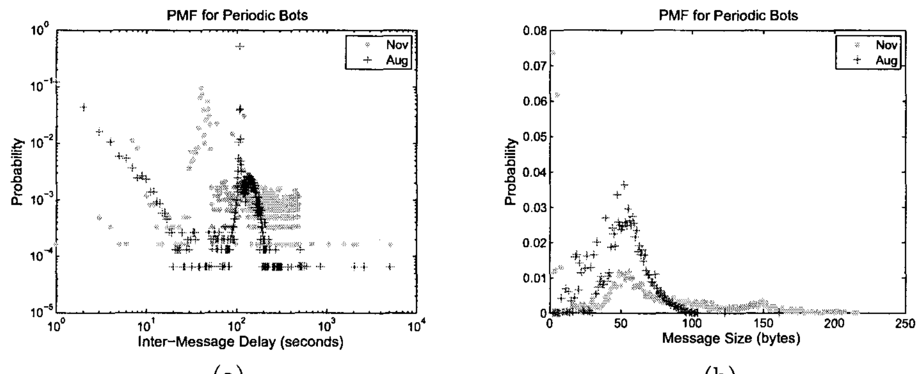


Figure 4.2: Distribution of Periodic Bot Inter-Message Delay (a) and Message Size (b)

In other words, the distribution is heavy tailed. We also find that the pmf curve of human message size in Figure 4.1 (b) can be well fitted by an exponential distribution with $\lambda = 0.034$ after excluding the initial spike.

4.2.2.2 Periodic Bots

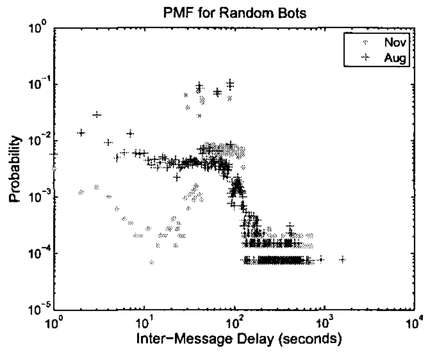
A periodic bot posts messages mainly at regular time intervals. The delay periods of periodic bots, especially those bots that use long delays, may vary by several seconds. The variation of delay period may be attributed to either transmission delay caused by network traffic congestion or chat server delay, or message emission delay incurred by system overloading on the bot hosting machine. The posting of periodic messages is a simple but effective mechanism for distributing messages, so it is not surprising that a substantial portion of chat bots use periodic timers.

We display the probability distributions of inter-message delay and message size for periodic bots in Figure 4.2. We use '+' for displaying August data and '•' for

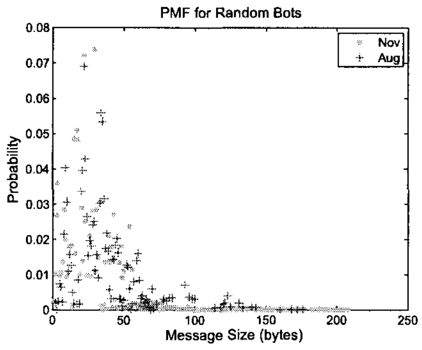
November data. The distributions of periodic bots are distinct from those of humans shown in Figure 4.1. The distribution of inter-message delay for periodic bots clearly manifests the timer-triggering characteristic of periodic bots. There are three clusters with high probabilities at time ranges [30-50], [100-110], and [150-170]. These clusters correspond to the November periodic bots with timer values around 40 seconds and the August periodic bots with timer values around 105 and 160 seconds, respectively. The message size pmf curve of the August periodic bots shows an interesting bell shape, much like a normal distribution. After examining message contents, we find that the bell shape may be attributed to the message composition method some August bots used. As shown in Appendix A, some August periodic bots compose a message using a single template. The template has several parts and each part is associated with several synonym phrases. Since the length of each part is independent and identically distributed, the length of whole message, i.e., the sum of all parts, should approximate a normal distribution. The November bots employ a similar composition method, but use several templates of different lengths. Thus, the message size distribution of the November periodic bots reflects the distribution of the lengths of the different templates, with the length of each individual template approximating a normal distribution.

4.2.2.3 Random Bots

A random bot posts messages at random time intervals. The random bots in our data used different random distributions, some discrete and others continuous, to generate



(a)

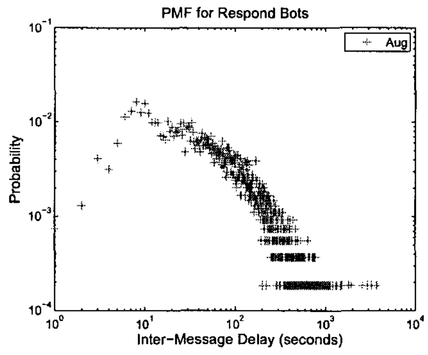


(b)

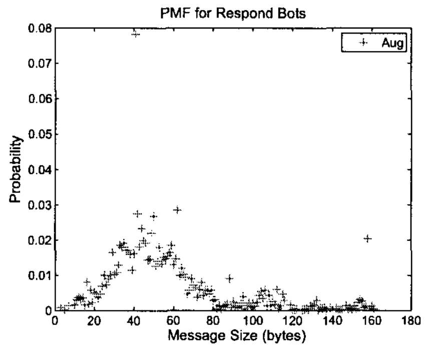
Figure 4.3: Distribution of Random Bot Inter-Message Delay (a) and Message Size (b)

inter-message delays. The use of random timers makes random bots appear more human-like than periodic bots. In statistical terms, however, random bots exhibit quite different inter-message delay distributions than humans.

Figure 4.3 depicts the probability distributions of inter-message delay and message size for random bots. Compared to periodic bots, random bots have more dispersed timer values. In addition, the August random bots have a large overlap with the November random bots. The points with high probabilities (greater than 10^{-2}) in the time range [30-90] in Figure 4.3 (a) represent the August and November random bots that use a discrete distribution of 40, 64, and 88 seconds. The wide November cluster with medium probabilities in the time range [40-130] is created by the November random bots that use a uniform distribution between 45 and 125 seconds. The probabilities of different message sizes for the August and November random bots are mainly in the size range [0-50]. Unlike periodic bots, most random bots do not use template or synonym replacement, but directly repeat messages. Thus, as their mes-

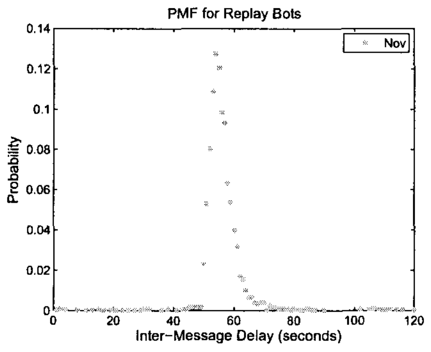


(a)

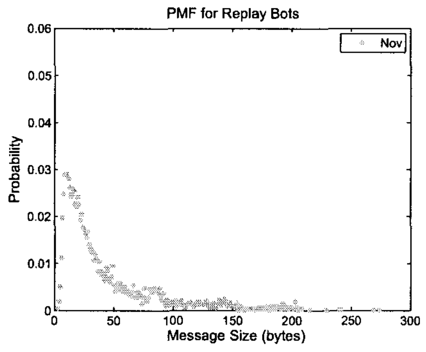


(b)

Figure 4.4: Distribution of Responder Bot Inter-Message Delay (a) and Message Size (b)



(a)



(b)

Figure 4.5: Distribution of Replay Bot Inter-Message Delay (a) and Message Size (b)

sages are selected from a database at random, the message size distribution reflects the proportion of messages of different sizes in the database.

4.2.2.4 Responder Bots

A responder bot sends messages based on the content of messages in the chat room. For example, a message ending with a question mark may trigger a responder bot to send a vague response with a URL, as shown in Appendix A. The vague response, in the context, may trick human users into believing that the responder is a human and further clicking the link. Moreover, the message triggering mechanism makes responder bots look more like humans in terms of timing statistics than periodic or random bots.

To gain more insights into responder bots, we managed to obtain a configuration file for a typical responder bot [119]. There are a number of parameters for making the responder bot mimic humans. The bot can be configured with a fixed typing rate, so that responses with different lengths take different time to "type." The bot can also be set to either ignore triggers while simulating typing, or rate-limit responses. In addition, responses can be assigned with probabilities, so that the responder bot responds to a given trigger in a random manner.

Figure 4.4 shows the probability distributions of inter-message delay and message size for responder bots. Note that only the distribution of the August responder bots is shown due to the small number of responder bots found in November. Since the message emission of responder bots is triggered by human messages, theoretically the distribution of inter-message delays of responder bots should demonstrate certain similarity to that of humans. Figure 4.4 (a) confirms this hypothesis. Like Figure 4.1 (a), the pmf of responder bots (excluding the head part) in log-log scale exhibits a clear sign of a heavy tail. But unlike human messages, the sizes of responder bot messages vary in a much narrower range (between 1 and 160). The bell shape of the distribution for message size less than 100 indicates that responder bots share a similar message composition technique with periodic bots, and their messages are composed as templates with multiple parts, as shown in Appendix A.

4.2.2.5 Replay Bots

A replay bot not only sends its own messages, but also repeats messages from other users to appear more like a human user. In our experience, replayed phrases are related to the same topic but do not appear in the same chat room as the original ones. Therefore, replayed phrases are either taken from other chat rooms on the same topic or saved previously in a database and replayed.

The use of replayed phrases in a crowded or "noisy" chat room does, in fact, make replay bots look more like human to inattentive users. The replayed phrases are sometimes nonsensical in the context of the chat, but human users tend to naturally ignore such statements. When replay bots succeed in fooling human users, these users are more likely to click links posted by the bots or visit their profiles. Interestingly, replay bots sometimes replay phrases uttered by other chat bots, making them very easy to be recognized. The use of replay is potentially effective in thwarting detection methods, as detection tests must deal with a combination of human and bots phrases. By using human phrases, replay bots can easily defeat keyword-based message filters that filter message-by-message, as the human phrases should not be filtered out.

Figure 4.5 illustrates the probability distributions of inter-message delay and message size for replay bots. In terms of inter-message delay, a replay bot is just a variation of a periodic bot, which is demonstrated by the high spike in Figure 4.5 (a). By using human phrases, replay bots successfully mimic human users in terms

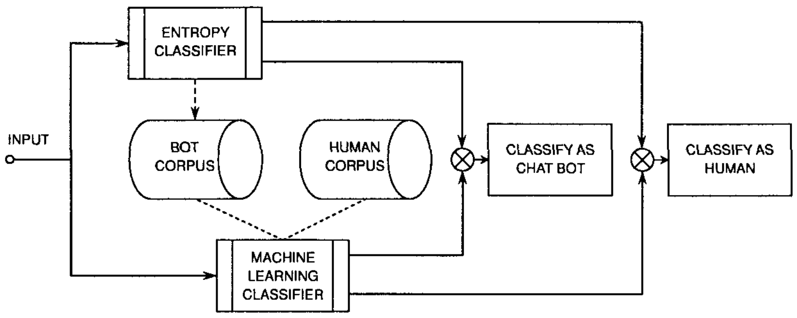


Figure 4.6: Classification System Diagram

of message size distribution. The message size distribution of replay bots in Figure 4.5 (b) largely resembles that of human users, and can be fitted by an exponential distribution with $\lambda = 0.028$.

4.3 Classification System

This section describes the design of our chat bot classification system. The two main components of our classification system are the entropy classifier and the machine learning classifier. The basic structure of our chat bot classification system is shown in Figure 4.6. The two classifiers, entropy and machine learning, operate concurrently to process input and make classification decisions, while the machine learning classifier relies on the entropy classifier to build the bot corpus. The entropy classifier uses entropy and corrected conditional entropy to score chat users and then classifies them as chat bots or humans. The main task of the entropy classifier is to capture new chat bots and add them to the chat bot corpus. The human corpus can be taken from a database of clean chat logs or created by manual log-based classification, as described in Section 5.2. The machine learning classifier uses the bot and human corpora to learn text patterns of bots and humans, and then it can quickly classify chat bots based on these patterns. The two classifiers are detailed as follows.

4.3.1 Entropy Classifier

The entropy classifier makes classification decisions based on entropy and entropy rate measures of message sizes and inter-message delays for chat users. If either the entropy or entropy rate is low for these characteristics, it indicates the regular or predictable behavior of a likely chat bot. If both the entropy and entropy rate is high for these characteristics, it indicates the irregular or unpredictable behavior of a possible human.

To use entropy measures for classification, we set a cutoff score for each entropy measure. If a test score is greater than or equal to the cutoff score, the chat user is classified as a human. If the test score is less than the cutoff score, the chat user is classified as a chat bot. The specific cutoff score is an important parameter in determining the false positive and true positive rates of the entropy classifier. On the one hand, if the cutoff score is too high, then too many humans will be misclassified as bots. On the other hand, if the cutoff score is too low, then too many chat bots will be misclassified as humans. Due to the importance of achieving a low false positive rate, we select the cutoff scores based on human entropy scores to achieve a targeted false positive rate. The specific cutoff scores and targeted false positive rates are described in Section 5.4.

4.3.1.1 Entropy Measures

The entropy rate, which is the average entropy per random variable, can be used as a measure of complexity or regularity [101, 105, 44]. The entropy rate is defined as the conditional entropy of a sequence of infinite length. The entropy rate is upper-bounded by the entropy of the first-order probability density function or first-order entropy. A independent and identically distributed (i.i.d.) process has an entropy rate equal to its first-order entropy. A highly complex process has a high entropy rate, while a highly regular process has a low entropy rate.

A random process $X = \{X_i\}$ is defined as an indexed sequence of random variables. To give the definition of the entropy rate of a random process, we first define the entropy of a sequence of random variables as:

$$H(X_1, ..., X_m) = - \sum_{X_1,...,X_m} P(x_1, ..., x_m) \log P(x_1, ..., x_m),$$

where $P(x_1, ..., x_m)$ is the joint probability $P(X_1 = x_1, ..., X_m = x_m)$.

Then, from the entropy of a sequence of random variables, we define the conditional entropy of a random variable given a previous sequence of random variables as:

$$H(X_m \mid X_1, ..., X_{m-1}) = H(X_1, ..., X_m) - H(X_1, ..., X_{m-1}).$$

Lastly, the entropy rate of a random process is defined as:

$$\overline{H}(X) = \lim_{m \to \infty} H(X_m \mid X_1, ..., X_{m-1}).$$

Since the entropy rate is the conditional entropy of a sequence of infinite length, it cannot be measure for finite samples. Thus, we estimate the entropy rate with the conditional entropy of finite samples. In practice, we replace probability density functions with empirical probability density functions based on the method of histograms. The data is binned in Q bins of approximately equal probability. The empirical probability density functions are determined by the proportions of bin number sequences in the data, i.e., the proportion of a sequence is the probability of that sequence. The estimates of the entropy and conditional entropy, based on empirical probability density functions, are represented as: EN and CE, respectively.

There is a problem with the estimation of $CE(X_m \mid X_1, ..., X_{m-1})$ for some values of m. The conditional entropy tends to zero as m increases, due to limited data. If a specific sequence of length $m-1$ is found only once in the data, then the extension of this sequence to length m will also be found only once. Therefore, the length m sequence can be predicted by the length $m-1$ sequence, and the length m and $m-1$ sequences cancel out. If no sequence of length m is repeated in the data, then $CE(X_m \mid X_1, ..., X_{m-1})$ is zero, even for i.i.d. processes.

To solve the problem of limited data, without fixing the length of m, we use the corrected conditional entropy [101] represented as CCE. The corrected conditional entropy is defined as:

$$CCE(X_m \mid X_1, ..., X_{m-1}) = CE(X_m \mid X_1, ..., X_{m-1}) + perc(X_m) \cdot EN(X_1),$$

where $perc(X_m)$ is the percentage of unique sequences of length m and $EN(X_1)$ is the entropy with m fixed at 1 or the first-order entropy.

Table 4.1: Message Composition of Chat Bot and Human Datasets

	AUG. BOTS			NOV. BOTS			HUMANS
	periodic	random	responder	periodic	random	replay	human
# of msgs.	25,258	13,998	6,160	10,639	22,820	8,054	342,696

The estimate of the entropy rate is the minimum of the corrected conditional entropy over different values of m. The minimum of the corrected conditional entropy is considered to be the best estimate of the entropy rate from the available data.

4.3.2 Machine Learning Classifier

The machine learning classifier uses the content of chat messages to identify chat bots. Since chat messages (including emoticons) are text, the identification of chat bots can be perfectly fitted into the domain of machine learning text classification. Within the machine learning paradigm, the text classification problem can be formalized as $f : T \times C \rightarrow \{0, 1\}$, where f is the classifier, $T = \{t_1, t_2, ..., t_n\}$ is the texts to be classified, and $C = \{c_1, c_2, ..., c_k\}$ is the set of pre-defined classes [108]. Value 1 for $f(t_i, c_j)$ indicates that text t_i is in class c_j and value 0 indicates the opposite decision. There are many techniques that can be used for text classification, such as naïve Bayes, support vector machines, and decision trees. Among them, Bayesian classifiers have been very successful in text classification, particularly in email spam detection. Due to the similarity between chat spam and email spam, we choose Bayesian classification for our machine learning classifier for detecting chat bots. We leave study on the applicability of other types of machine learning classifiers to our future work.

Within the framework of Bayesian classification, identifying if chat message M is issued by a bot or human is achieved by computing the probability of M being from a bot with the given message content, i.e., $P(C = bot|M)$. If the probability is equal to or greater than a pre-defined threshold, then message M is classified as a bot message. According to Bayes theorem,

$$P(bot|M) = \frac{P(M|bot)P(bot)}{P(M)} = \frac{P(M|bot)P(bot)}{P(M|bot)P(bot) + P(M|human)P(human)}.$$

A message M is described by its feature vector $\langle f_1, f_2, ..., f_n \rangle$. A feature f is a single word or a combination of multiple words in the message. To simplify computation, in practice it is usually assumed that all features are conditionally independent with each other for the given category. Thus, we have

$$P(bot|M) = \frac{P(bot) \prod_{i=1}^{n} P(f_i|bot)}{P(bot) \prod_{i=1}^{n} P(f_i|bot) + P(human) \prod_{i=1}^{n} P(f_i|human)}.$$

The value of $P(bot|M)$ may vary in different implementations (see [51, 137] for implementation details) of Bayesian classification due to differences in assumption and simplification.

Given the abundance of implementations of Bayesian classification, we directly adopt one implementation, namely CRM 114[135], as our machine learning classification component. CRM 114 is a powerful text classification system that has achieved very high accuracy in email spam identification. The default classifier of CRM 114, OSB (Orthogonal Sparse Bigram), is a type of Bayesian classifier. Different from common Bayesian classifiers which treat individual words as features, OSB uses word pairs as features instead. OSB first chops the whole input into multiple basic units with five consecutive words in each unit. Then, it extracts four word pairs from each unit to construct features, and derives their probabilities. Finally, OSB applies Bayes theorem to compute the overall probability that the text belongs to one class or another.

4.4 Experimental Evaluation

In this section, we evaluate the effectiveness of our proposed classification system. Our classification tests are based on chat logs collected from the Yahoo! chat system. We test the two classifiers, entropy-based and machine-learning-based, against chat bots from August and November datasets. The machine learning classifier is tested with fully-supervised training and entropy-classifier-based training. The accuracy of classification is measured in terms of false positive and false negative rates. The false positives are those human users that are misclassified as chat bots, while the false negatives are those chat bots that are misclassified as human users. The speed of classification is mainly determined by the minimum number of messages that are required for accurate classification. In general, a high number means slow classification, whereas a low number means fast classification.

Table 4.2: True Positive and Negative Rates for Entropy Classifier

	AUG. BOTS			NOV. BOTS			HUMANS
	periodic	random	responder	periodic	random	replay	human
test	true pos.	true pos.	true pos.	true pos.	true pos.	true pos.	false pos.
EN(imd)	121/121	68/68	1/30	51/51	109/109	40/40	7/1713
CCE(imd)	121/121	49/68	4/30	51/51	109/109	40/40	11/1713
EN(ms)	92/121	7/68	8/30	46/51	34/109	0/40	7/1713
CCE(ms)	77/121	8/68	30/30	51/51	6/109	0/40	11/1713
OVERALL	121/121	68/68	30/30	51/51	109/109	40/40	17/1713

4.4.1 Experimental Setup

The chat logs used in our experiments are mainly in three datasets: (1) human chat logs from August 2007, (2) bot chat logs from August 2007, and (3) bot chat logs from November 2007. In total, these chat logs contain 342,696 human messages and 87,049 bot messages. In our experiments, we use the first half of each chat log, human and bot, for training our classifiers and the second half for testing our classifiers. The composition of the chat logs for the three datasets is listed in Table 4.1.

The entropy classifier only requires a human training set. We use the human training set to determine the cutoff scores, which are used by the entropy classifier to decide whether a test sample is a human or bot. The target false positive rate is set at 0.01. To achieve this false positive rate, the cutoff scores are set at approximately the 1st percentile of human training set scores. Then, samples that score higher than the cutoff are classified as humans, while samples that score lower than the cutoff are classified as bots. The entropy classifier uses two entropy tests: entropy and corrected conditional entropy. The entropy test estimates first-order entropy, and the corrected conditional entropy estimates higher-order entropy or entropy rate. The corrected conditional entropy test is more precise with coarse-grain bins, whereas the entropy test is more accurate with fine-grains bins [44]. Therefore, we use $Q = 5$ for the corrected conditional entropy test and $Q = 256$ with m fixed at 1 for the entropy test.

We run classification tests for each bot type using the entropy classifier and machine learning classifier. The machine learning classifier is tested based on fully-supervised training and then entropy-based training. In fully-supervised training, the machine learning classifier is trained with manually labeled data, as described in Section 5.2. In entropy-based training, the machine learning classifier is trained with data labeled by the entropy classifier. For each evaluation, the entropy classifier uses samples of 100 messages, while the machine learning classifier uses samples of 25 messages.

4.4.2 Experimental Results

We now present the results for the entropy classifier and machine learning classifier. The four chat bot types are: periodic, random, responder, and replay. The classification tests are organized by chat bot type, and are ordered by increasing detection difficulty.

4.4.2.1 Entropy Classifier

The detection results of the entropy classifier are listed in Table 4.2, which includes the results of the entropy test (EN) and corrected conditional entropy test (CCE) for inter-message delay (imd), and message size (ms). The overall results for all entropy-based tests are shown in the final row of the table. The true positives are the total unique bot samples correctly classified as bots. The false positives are the total unique human samples mistakenly classified as bots.

Periodic Bots: As the simplest group of bots, periodic bots are the easiest to detect. They use different fixed timers and repeatedly post messages at regular intervals. Therefore, their inter-message delays are concentrated in a narrower range than those of humans, resulting in lower entropy than that of humans. The inter-message delay EN and CCE tests detect 100% of all periodic bots in both August and November datasets. The message size EN and CCE tests detect 76% and 63% of the August periodic bots, respectively, and 90% and 100% of the November periodic bots, respectively. These slightly lower detection rates are due to a small proportion of humans with low entropy scores that overlap with some periodic bots. These humans post mainly short messages, resulting in message size distributions with low entropy.

Random Bots: The random bots use random timers with different distributions. Some random bots use discrete timings, e.g., 40, 64, or 88 seconds, while the others use continuous timings, e.g., uniformly distributed delays between 45 and 125 seconds.

The inter-message delay EN and CCE tests detect 100% of all random bots, with one exception: the inter-message delay CCE test against the August random bots only achieves 72% detection rate, which is caused by the following two conditions: (1) the range of message delays of random bots is close to that of humans; (2) sometimes the randomly-generated delay sequences have similar entropy rate to human patterns. The message size EN and CCE tests detect 31% and 6% of August random bots, respectively, and 7% and 8% of November random bots, respectively. These low detection rates are again due to a small proportion of humans with low message size entropy scores. However, unlike periodic bots, the message size distribution of random bots is highly dispersed, and thus, a larger proportion of random bots have high entropy scores, which overlap with those of humans.

Responder Bots: The responder bots are among the advanced bots, and they behave more like humans than random or periodic bots. They are triggered to post messages by certain human phrases. As a result, their timings are quite similar to those of humans.

The inter-message delay EN and CCE tests detect very few responder bots, only 3% and 13%, respectively. This demonstrates that human-message-triggered responding is a simple yet very effective mechanism for imitating the timing of human interactions. However, the detection rate for the message size EN test is slightly better at 27%, and the detection rate for the message size CCE test reaches 100%. While the message size distribution has sufficiently high entropy to frequently evade the EN test, there is some dependence between subsequent message sizes, and thus, the CCE detects the low entropy pattern over time.

Replay Bots: The replay bots also belong to the advanced and human-like bots. They use replay attacks to fool humans. More specifically, the bots replay phrases they observed in chat rooms. Although not sophisticated in terms of implementation, the replay bots are quite effective in deceiving humans as well as frustrating our message-size-based detections: the message size EN and CCE tests both have detection rates of 0%. Despite their clever trick, the timing of replay bots is periodic and easily detected. The inter-message delay EN and CCE tests are very successful at detecting replay bots, both with 100% detection accuracy.

4.4.2.2 Supervised and Hybrid Machine Learning Classifiers

The detection results of the machine learning classifier are listed in Table 4.3. Table 4.3 shows the results for the fully-supervised machine learning ($SupML$) classifier

Table 4.3: True Positive and Negative Rates for Machine Learning and Hybrid Classifiers

	AUG. BOTS			NOV. BOTS			HUMANS
	periodic	random	responder	periodic	random	replay	human
test	true pos.	true pos.	true pos.	true pos.	true pos.	true pos.	false pos.
SupML	121/121	68/68	30/30	14/51	104/109	1/40	0/1713
SupMLretrained	121/121	68/68	30/30	51/51	109/109	40/40	0/1713
EntML	121/121	68/68	30/30	51/51	109/109	40/40	1/1713

and entropy-trained machine learning (*EntML*) classifier, both trained on the August training datasets, and the fully-supervised machine learning (*SupMLretrained*) classifier trained on August and November training datasets.

Periodic Bots: For the August dataset, both *SupML* and *EntML* classifiers detect 100% of all periodic bots. For the November dataset, however, the *SupML* classifier only detects 27% of all periodic bots. The lower detection rate is due to the fact that 62% of the periodic bot messages in November chat logs are generated by new bots, making the *SupML* classifier ineffective without re-training. The *SupMLretrained* classifier detects 100% of November periodic bots. The *EntML* classifier also achieves 100% for the November dataset.

Random Bots: For the August dataset, both *SupML* and *EntML* classifiers detect 100% of all random bots. For the November dataset, the *SupML* classifier detects 95% of all random bots, and the *SupMLretrained* classifier detects 100% of all random bots. Although 52% of the random bots have been upgraded according to our observation, the old training set is still mostly effective because certain content features of August random bots still appear in November. The *EntML* classifier again achieves 100% detection accuracy for the November dataset.

Responder Bots: We only present the detection results of responder bots for the August dataset, as the number of responder bots in the November dataset is very small. Although responder bots effectively mimic human timing, their message contents are only slightly obfuscated and are easily detected. The *SupML* and *EntML* classifiers both detect 100% of all responder bots.

Replay Bots: The replay bots only exist in the November dataset. The *SupML* classifier detects only 3% of all replay bots, as these bots are newly introduced in November. The *SupMLretrained* classifier detects 100% of all replay bots. The machine learning classifier reliably detects replay bots in the presence of a substantial number of replayed human phrases, indicating the effectiveness of machine learning techniques in chat bot classification.

4.5 Conclusion

This chapter first presents a large-scale measurement study on Internet chat. We collected two-month chat logs for 21 different chat rooms from one of the top Internet chat service providers. From the chat logs, we identified a total of 14 different types of chat bots and grouped them into four categories: periodic bots, random bots, responder bots, and replay bots. Through statistical analysis on inter-message delay and message size for both chat bots and humans, we found that chat bots behave very differently from human users. More specifically, chat bots exhibit certain regularities in either inter-message delay or message size. Although responder bots and replay bots employ advanced techniques to behave more human-like in some aspects, they still lack the overall sophistication of humans.

Based on the measurement study, we further proposed a chat bot classification system, which utilizes entropy-based and machine-learning-based classifiers to accurately detect chat bots. The entropy-based classifier exploits the low entropy characteristic of chat bots in either inter-message delay or message size, while the machine-learning-based classifier leverages the message content difference between humans and chat bots. The entropy-based classifier is able to detect unknown bots, including human-like bots such as responder and replay bots. However, it takes a relatively long time for detection, i.e., a large number of messages are required. Compared to the entropy-based classifier, the machine-learning-based classifier is much faster, i.e., a small number of messages are required. In addition to bot detection, a major task of the entropy-based classifier is to build and maintain the bot corpus. With the help of bot corpus, the machine-learning-based classifier is trained, and consequently, is able to detect chat bots quickly and accurately. Our experimental results demonstrate that the hybrid classification system is fast in detecting known bots and is accurate in identifying previously-unknown bots.

Chapter 5

Detecting Online Game Bots

The online gaming market has experienced rapid growth for the past few years. In 2008, online gaming revenues were estimated at \$7.6 billion world-wide [81]. The most profitable online games are subscription-based massive multiplayer online games (MMOGs), such as World of Warcraft. In 2008, World of Warcraft reached 11.5 million subscribers [16]. Each subscriber has to pay as much as \$15 per month. It is no surprise that MMOGs make up about half of online gaming revenues [81]. As MMOGs gain in economic and social importance, it has become imperative to shield MMOGs from malicious exploits for the benefit of on-line game companies and players.

Currently the most common form of malicious exploit and the most difficult to thwart, is the use of game bots to gain unfair advantages. Game bots have plagued most of the popular MMOGs, including World of Warcraft [80, 104, 129, 100, 95], Second Life [92], and Ultima Online [40, 113], and some non-MMOGs such as Diablo 2 [33]. The primary goal of game bots is to amass game currency, items, and experience.

Interestingly, game currency can be traded for real currency[1], making cheating a profitable enterprise. Since MMOGs are small economies, a large influx of game currency causes hyper-inflation, hurting all players. Thus, the use of game bots is a serious problem for not only giving some players unfair advantages but also for creating large imbalances in game economies as a whole. With a large investment in development costs, game service providers consider anti-cheating mechanisms a high priority.

The existing methods for combating bots are not successful in the protection of on-line games. The approaches based on human interactive proofs (HIPs), such as CAPTCHAs, are the most commonly used to distinguish bots from humans. However, the inherent interactive requirement makes HIP-based approaches inadequate to apply in MMOGs. In particular, multiple tests are needed throughout a game session to block the login of bots; otherwise, a malicious player can pass the one-time test and log a bot into the game. Although multiple tests can foil the malicious player's attempt for bot login, they are too obtrusive and distractive for a regular player to tolerate as well. A different approach, taken by some game companies, makes use of a process monitor to scan for known bot or cheat programs running on a player's computer. Blizzard, the makers of World of Warcraft, developed such a system called the Warden that scans processes and sends information back to their servers. A number of similar anti-cheat systems have been built for other games [39, 120, 94, 35]. However, this scan-based approach has proven ineffective, and even worse, raises privacy concerns.

[1]The exchange rate for World of Warcraft is 1,000 gold to $11.70 as of July 25th, 2009 [117].

The Electronic Frontier Foundation views the Warden as spyware [79].

Besides technical approaches, Blizzard has pursued legal action against bot makers [9], claiming over $1 million per year in additional operating costs caused by game bots in their lawsuit [15]. Moreover, Blizzard has banned thousands of accounts for cheating [21], yet many players continue cheating via bots and slip through the cracks [95, 100].

In this chapter, we introduce an approach based on human observational proofs (HOPs) to capture game bots. HOPs offer two distinct advantages over HIPs. First, HOPs provide continuous monitoring throughout a session. Second, HOPs are non-interactive, i.e., no test is presented to a player, making HOPs completely non-obtrusive. The use of HOPs is mainly motivated by the problems faced by HIPs and methods used in behavioral biometric systems [96, 103, 43, 2]. Similar behavior-based approaches have been used in many previous intrusion detection systems [46, ?, 67, 102, 121]. We collect a series of user-input measurements from a popular MMOG, World of Warcraft, to study the behaviors of current game bots and humans. While human players visually recognize objects on the screen and physically control the mouse and keyboard, game bots synthetically generate mouse and keyboard events and cannot directly recognize most objects. Our measurement results clearly show the fundamental differences between current game bots and humans in how certain tasks are performed in the game. Passively observing these differences, HOPs provide an effective way to detect current game bots.

Based on HOPs, we design and develop a game bot defense system that analyzes user-input data to differentiate game bots from human players in a timely manner. The proposed HOP system consists of two major components: a client-side exporter and a server-side analyzer. The exporter is responsible for sending a stream of user-input actions to the server. The analyzer then processes the user-input stream and decides whether the client is operated by a bot or a human. The core of the analyzer is a cascade neural network that "learns" the behaviors of normal human players, as neural networks are known to perform well with user-input data [2, 90, 91]. Note that the latest MMOGs virtually all support automatic updates, so the deployment of the client-side exporter is not an issue. Moreover, the overhead at the client side is negligible and the overhead at the server side is small and affordable in terms of CPU and memory consumptions even with thousands of players per server. To validate the efficacy of our defense system, we conduct experiments based on user-input traces of bots and humans. The HOP system is able to capture 99.80% of current game bots for World of Warcraft within 39.60 seconds on average.

It is an arms race between game exploits and their countermeasures. Once highly motivated bot developers know the HOP approach, it is possible for them to create more advanced game bots to evade the HOP system. However, the purpose of the HOP system is to raise the bar against game exploits and force a determined bot developer to spend significant time and effort in building next-generation game bots for detection evasion. Note that, to operate the game in a human-like manner, game bots have to process complex visuals and model different aspects of human-computer interaction and behavior, which we believe is non-trivial to succeed.

5.1 Background

In this section, we first briefly present the evolution of game bots. Then, we describe the game playing behaviors of human players and game bots, respectively, and highlight their differences in a qualitative way.

5.1.1 Game Bots

A variety of exploits have appeared in the virtual game world for fun, for win, and for profit. Among these game exploits, game bots are regarded as the most commonly-used and difficult-to-handle exploit. The earliest game bots were developed for the first generation MMOGs such as Ultima Online [113]. Even at that time, bot operators were already quite sophisticated, creating small server farms to run their bots [40, 113]. At the early era of game bots, most of bot programmers wrote their own game clients. However, as a countermeasure, game companies often update games, breaking operations of those custom game clients. Bot programmers were forced to update their game clients, keeping up with the latest game version. This cycle proves to be very tedious for game bot programmers. Moreover, the complexity of game clients has grown continuously, making it increasingly difficult to develop and maintain a standalone custom game client.

The arms race between game vendor and bot developer has led to the birth of an interesting type of game bots that, much like humans, play games by reading from screen and using the mouse and keyboard. These advanced bots operate the standard game client by simply sending mouse and keyboard events, reading certain pixels from the screen, and possibly reading a few key regions in the memory address space of the game application. Most bots are equipped with macro scripting capabilities, similar to programs like AutoIt [10], which enables bots to be easily reprogrammed and quickly adapted to the changes made by game companies.

5.1.2 Game Playing Behaviors

MMOGs, such as World of Warcraft, entertain players by providing a large degree of freedom in terms of actions a player can perform. In the game world, a player controls a virtual character (avatar) to explore the landscape, fight monsters, complete quests and interact with other players. In addition, a player can further customize the character by learning skills and purchasing items (such as armor, weapons, and even pets) with virtual currency. Each game activity requires a player to interact with the game in a different fashion. As a result, it is expected that the inputs of a human player will exhibit burstiness with strong locality and the input contents vary significantly for different tasks through game play. However, when a bot is used to play the game, its main purpose is to gain rewards (level and virtual currency) without human intervention by automating and repeating simple actions (such as killing monsters). Being much less sophisticated than human, bot actions would show regular patterns and limited varieties.

Besides the high-level behavioral differences, humans and bots also interact with the game very differently, despite that both interact with the game via mouse and keyboard. As biological entities, humans perceive the graphical output of the game optically, and feed input to the game by physically operating devices such as keyboard and mouse. In contrast, bots are computer programs that have no concept of vision and are not bounded by mechanical physics. While bots can analyze game graphics, it is computationally expensive. To avoid this computation cost, whenever possible, bots attempt to obtain necessary information, such as the locations of the avatar, monsters and other characters, and the properties (health, level, etc.) of the avatar, by reading the memory of the game program.

In general, bots control the avatar by simulating input from devices via OS API calls, such as setting key press state or repositioning mouse cursor. The techniques used by bots are often crude, but in most cases, quite effective. For example, without reading the graphics or scanning the terrain, a bot can navigate to a target location by knowing just two coordinates—the current location of the avatar and that of the target. The bot then tries to approach the target location by steering the avatar to go forward, left and right, and then checks its progress by polling the two coordinates. If the avatar location does not change in a given amount of time, the bot assumes that an obstacle (trees, fences, steep terrain, etc.) is in the way and tries to navigate around it by moving backward a few steps, turning left or right, and going forward. Occasionally, graphics analysis can be useful, such as when picking up items on the ground. The bot can again handle this situation in a simple and efficient manner by exploiting the game user interface. When the cursor is placed on top of an object, the game would display a small information window on the lower-right corner. Thus, the bot moves the mouse cursor in grid patterns, and relies on the change of pixel colors on the lower-right corner of the screen to know if it has found the object.

5.2 Game Playing Characterization

In this section, we examine how bots and humans behave in the game, in order to have a deep understanding of the differences between humans and bots. Based on our game measurements, we quantitatively characterize the game playing behaviors of human players and bots, respectively. The behavioral differences between bots and humans form the basis for our HOP-based system.

5.2.1 The Glider Bot

We select the Glider bot [80] as the sample game bot for our research. The Glider bot is a very popular game bot for World of Warcraft. It runs concurrently with the game client, but requires system administrator privileges. This escalated privilege helps the Glider bot to circumvent the Warden anti-bot system, and enables it to access the internal information of the game client via cross-process-address-space reading. It operates by using a "profile"—a set of configurations including several waypoints (map coordinates in the game world) and options, such as levels of monsters to fight. When in operation, the game bot controls the avatar to repeatedly run between the given waypoints, search and fight monsters that match the given criteria, and collect bonus items after winning fights.

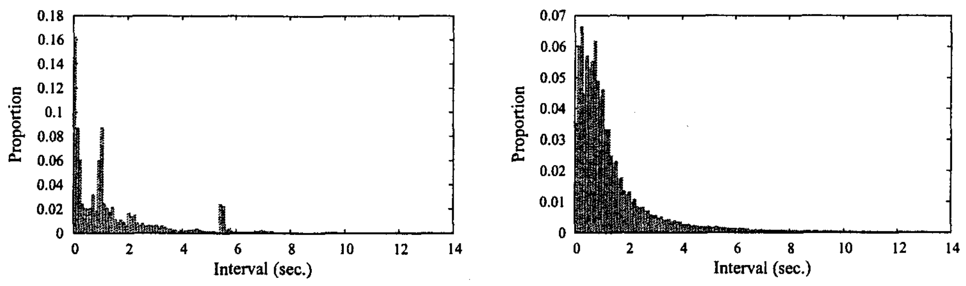
Table 5.1: Definitions of User-Input Actions

Action	Definition
Keystroke	The press and release of a key.
Point	A series of continuous mouse cursor position changes with no mouse button pressed; the time-stamps for each pair of cursor position changes are no more than 0.4 seconds apart.
Pause	A period of 0.4 seconds or longer with no actions.
Click	The press and release of a mouse button; the cursor travels no more than 10 pixels between the press and release.
Point-and-Click	A point followed by a click within 0.4 seconds.
Drag-and-Drop	The press and release of a mouse button; the cursor travels more than 10 pixels between the press and release.

5.2.2 Input Data Collection

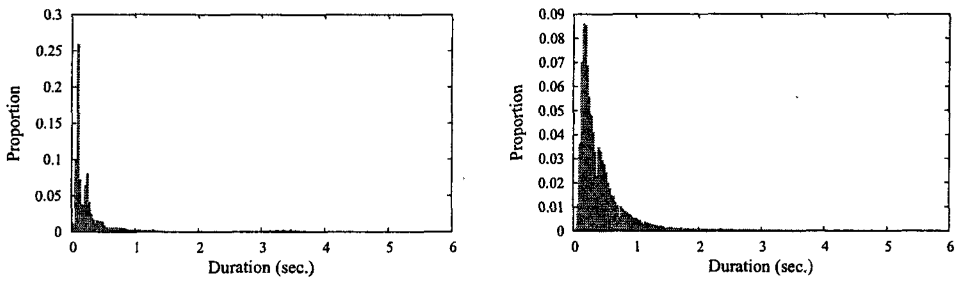
We collect player input data for both human and bot using an external program in a non-intrusive manner, i.e., no modification to the game client program. The input data collection program, a modified version of RUI [69], runs concurrently with the game, polling and recording the keyboard and mouse input device status with clock resolution close to 0.015625 second (approximate 64 times/sec). Each input event, such as key press or cursor position change, is recorded along with a time stamp relative to the starting time of the recording.

We invite 30 different human players to play World of Warcraft and collect 55 hours of their user-input traces. Correspondingly, we run the game bot with 10 different



(a) Bot (b) Human

Figure 5.1: Keystroke Inter-arrival Time Distribution

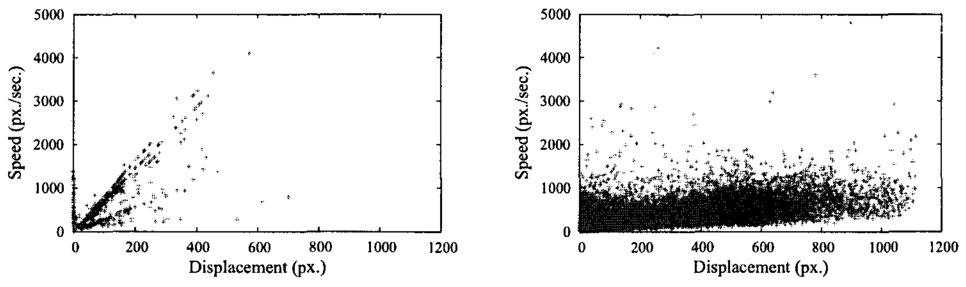


(a) Bot (b) Human

Figure 5.2: Keystroke Duration Distribution

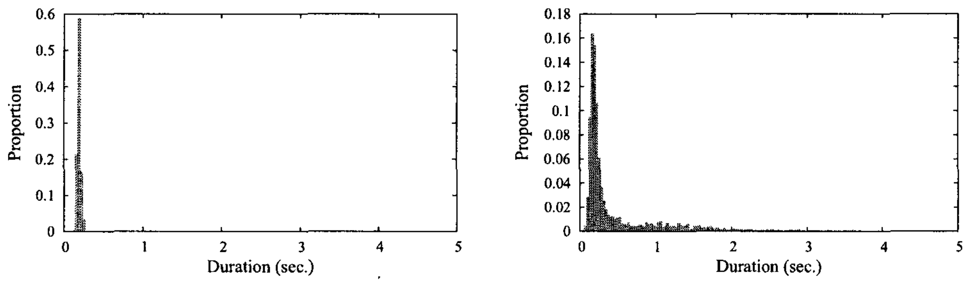
profiles in 7 locations in the game world for 40 hours and collect its input traces. The 10 profiles are bot configurations with different sets of waypoints that the bot follows while farming, i.e., killing monsters and gathering treasure. The profiles are setup in 7 locations with different monster levels (from levels 1 to 40), monster densities (sparse to dense), and different obstacles (barren plains to forest with lots of small trees). The game bot profiles are half run with a warrior and half run with a mage. These two bot characters range from level 1 to over 30 in the traces.

We conduct post processing on the input trace data to extract information with regard to high-level user-input actions. For example, we pair up a key press event with a subsequent key release event of the same key to form a `keystroke` action; we



(a) Bot (b) Human

Figure 5.3: Average Speed vs. Displacement for Point-and-Click



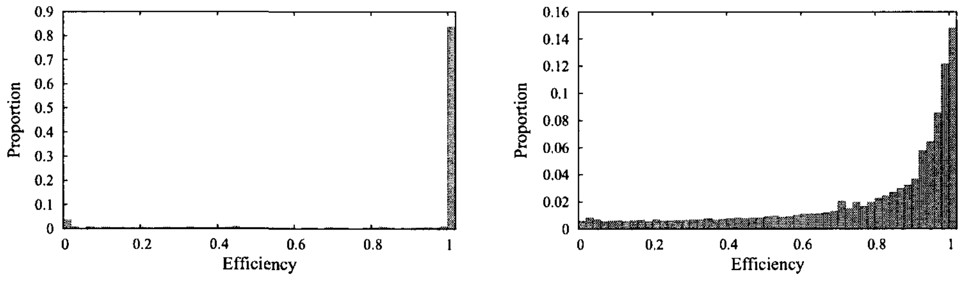
(a) Bot (b) Human

Figure 5.4: Drag-and-Drop Duration Distribution

gather a continuous sequence of cursor position change events to form a `point` action (mouse movement action). Table 5.1 gives a complete list of high level actions we derive and their corresponding definitions.

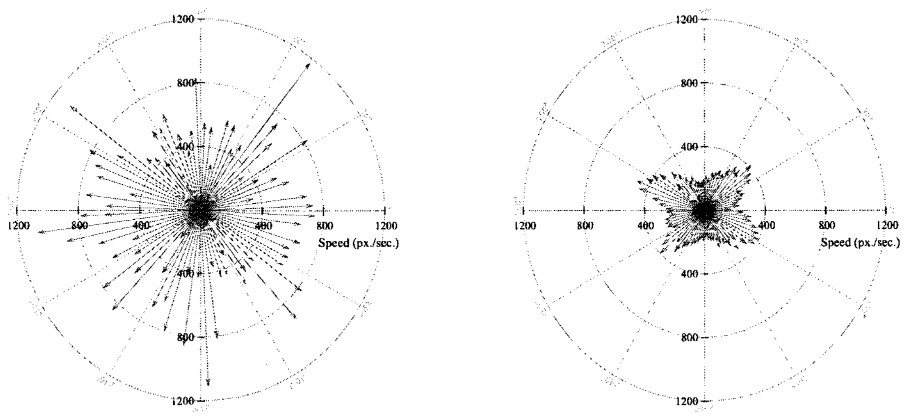
5.2.3 Game Playing Input Analysis

We analyze the Glider bot and human keyboard and mouse input traces with respect to timing patterns (duration and inter-arrival time) and kinematics (distance, displacement, and velocity). Our bot analysis below is limited to the current game bots.



(a) Bot (b) Human

Figure 5.5: Point-and-Click and Drag-and-Drop Movement Efficiency Distribution



(a) Bot (b) Human

Figure 5.6: Average Velocity for Point-and-Click

Two keyboard usage metrics for human and bot are presented in Figures 5.1 and 5.2, respectively. Both figures are clipped for better presentation, and the trailing data clipped away contribute less than 3% of the total for either human or bot. Figure 5.1 shows the distribution of `keystroke` inter-arrival time, i.e., the interval between two consecutive key presses, with a bin resolution of 0.1 seconds. There are two major differences between the bots and humans. First, the bot issues keystrokes significantly faster than humans. While 16.2% of consecutive keystrokes by the bot are less than 0.1 second apart, only 3.2% of human keystrokes are that fast. This is because human players have to initiate keystroke action by physical movement of fingers, and hence, pressing keys at such high frequency would be very tiring. Second, the keystrokes of the bot exhibit obvious periodic patterns. The empirical probabilities of the bot pressing a key every 1 or 5.5 seconds are significantly higher than their neighbor intervals, which provides us some insights into the internals of the bot: it uses periodic timers to poll the status of the avatar (i.e., current coordinate), and issue keyboard commands accordingly (e.g., bypass possible obstacles by turning left/right and jumping). However, for human players, their keystroke intervals follow a Pareto distribution, which matches the conclusions of previous research [127]. Figure 5.2 shows the distribution of `keystroke` durations, with the bin resolution of 0.03 second. These figures reassures our previous observations: the bot presses keys with much shorter duration—over 36.9% of keystrokes are less than 0.12 seconds long, while only 3.9% of human keystrokes are completed within such a duration; the bot exhibits the periodic keyboard usage pattern—keystrokes with around 0.25 second duration are significantly more than its neighbor durations.

Figure 5.3 shows the relationship between the mouse speed and the displacement between the origin and target coordinates for the `point-and-click`. Less than 0.1% of the total data points for either human or bot are clipped away. The bots exhibit two very unique features. First, unlike human players, who move the mouse with very dynamic speed at all displacement lengths, the bots tend to move the mouse at several fixed speeds for each displacement, and the speed increases linearly as displacement lengthens. This feature implies that, again, the bots use several fixed length timers for mouse movements. Second, we also observe that the bots make a significant amount of high speed moves with zero displacement, that is, after a series of fast movements, the cursor is placed back exactly at its origin. Such a behavior is absent in the human data, because it is physically difficult and unnecessary.

Figure 5.4 shows the distribution of mouse `drag-and-drop` duration, with the bin resolution of 0.03 second. For the bots, 100% of actions are accomplished within 0.3 second. However, for human players, only 56.6% of `drag-and-drop` actions finish within the same time window; over-one-second actions contribute 25.5% of the total, within which, about 0.8% of actions are more than 5 seconds long, and are thus clipped away from the figure.

Figure 5.5 illustrates the distribution of mouse movement efficiency for `point-and-click` and `drag-and-drop`. We define *movement efficiency* as the ratio between the cursor displacement and the traversed distance over a series of movements. In other words, the closer the cursor movement is to a straight line between the origin and target coordinates, the higher the movement efficiency. Note that, while the bin width is 0.02, the last bin only contains the actions with efficiency of 1.0. Bots exhibit significant deviation from human players on this metric: 81.7% of bot mouse movements have perfect efficiency, compared to that only 14.1% of human mouse movements are equally efficient. Aside from 3.8% of mouse movements with efficiency less than 0.02 (most of which are zero efficiency moves, due to the cursor being placed back to the origin), a bot rarely moves the mouse with other efficiencies. However, for human players, the observed probability of mouse movement efficiency follows an

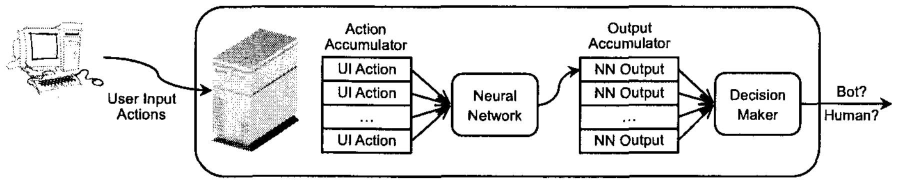


Figure 5.7: Overview of the HOP System

exponential distribution.

Finally, Figure 5.6 presents the relationship between the average mouse move speed and the direction of the target coordinate, plotted in polar coordinate with angular resolution of 10 degrees ($\pi/36$). Each arrow represents the average velocity vector of mouse movements whose target position is ±5 degrees in its direction. For the bots, there is no evident correlation between the speed and the direction. In contrast, for human players, there is a clear *diagonal, symmetric, and bounded* movement pattern: diagonal movements are generally faster than horizontal and vertical movements, upward movements are slightly faster than downward movements, and leftward movements are slightly faster than rightward movements; overall, the movement speed is bounded to a certain value. The diagonal and symmetric pattern is attributed to the human hand physiology, and the speed boundary is due to the physical constraint of human arms.

5.3 HOP System

In this section, we describe the design of our proposed HOP system. The HOP system consists of client-side exporters and a server-side analyzer. Each client-side exporter collects and sends a stream of user-input actions taken at a game client to the game server. The server-side analyzer then processes each input stream and decides whether the corresponding client is operated by a bot or a human player. Figure 5.7 illustrates the high-level structure of the HOP system.

5.3.1 Client-Side Exporter

Since each game client already receives raw user-input events, the client-side exporter simply uses the available information to derive input actions, i.e., `keystroke`, `point`, `click`, and `drag-and-drop`, and sends them back to the server along with regular game-related data. Ideally, the client-side exporter should be implemented as an integral part of the game executable or existing anti-cheat systems [39, 120, 94, 35]. For the prototype of our HOP system, we implement it as a standalone external program, as we do not have source code access to the World of Warcraft.

5.3.2 Server-Side Analyzer

The server-side analyzer is composed of two major components: the user-input action classifier and the decision maker. The work-flow of the server-side analyzer is as follows. For each user-input action stream, the system first stores consecutive actions into the action accumulator. A configurable number of actions form an action block, and each action block is then processed by the classifier. The output of the classifier contains the classification score for the corresponding action block, i.e., how close the group of actions look to those of a bot, and is stored into the output accumulator. Finally, when the output accumulator aggregates a configurable amount of neural network output, the decision maker makes a judgment. Each judgment reflects whether the player is possibly operated by a bot since the last judgment. The output accumulator is refreshed after each decision is made. The analyzer continuously processes user-input actions throughout each user's game session.

5.3.2.1 Neural Network Classification

We employ artificial neural networks for user-input action classification due to the following two reasons. First, neural networks are especially appropriate for solving pattern recognition and classification problem involving a large number of parameters with complex inter-dependencies. The effectiveness of neural networks with user-input data classification has already been demonstrated in behavioral biometric identification systems [2, 90, 91]. Second, neural networks are not simple functions of their inputs and outputs. While the detection methods based solely on those metrics with clearly defined equations are susceptible to inverse function attacks, neural networks, often described as a "black box", are more difficult to attack. Note that our HOP system is not necessarily tied to neural networks, and we will consider other classification methods, such as support vector machines (SVMs) or decision trees, in our future work.

The neural network we build for the HOP system is a cascade-correlation neural network, a variant of feed-forward neural networks that use the idea of cascade training [41]. Unlike standard multi-layer back-propagation (BP) perceptron networks, a cascade correlation neural network does not have a fixed topology, but rather is built from the ground up. Initially, the neural network only consists of the inputs directly connected to the output neuron. During the training of the neural network, a group of neurons are created and trained separately, and the best one is inserted into the network. The training process continues to include new neurons into the network, until the neural network reaches its training target or the size of the network reaches a pre-defined limit.

Figure 5.8 illustrates the general construction of the cascade-correlation neural network. There are eight input values for each user-input action, including seven action metric parameters and a bias value that is used to differentiate the type of action, e.g., keyboard action or mouse action. The neural network takes input from all actions in an action block. The connections between the input node and neurons, and among neurons, are represented by intersections between a horizontal line and a vertical line. The weight of each connection is shown as a square over the intersection, where larger size indicates heavier weight.

The seven action metric parameters are: action duration, mouse travel distance, displacement, efficiency, speed, angle of displacement, and virtual key (a numeric value corresponding to a keyboard key or a mouse button). The speed and efficiency are derived parameters from the basic parameters, such as duration, distance and displacement. These derived parameters are used mainly to help the neural network capture the inherent association between input parameters, reduce the network complexity, and thus, speedup the training process. The number of actions in an action block directly affects the total amount of input data to the neural network. Increasing the block size provides the neural network with more context information and can, up

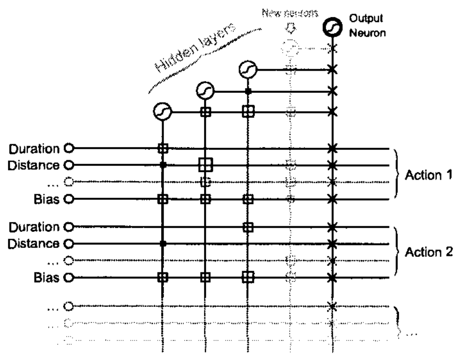


Figure 5.8: A Cascade Neural Network

to a certain point, further improve the classification accuracy of the trained network. However, too many input actions can also increase the overall complexity of the neural network and slow down the training process.

5.3.2.2 Decision Making

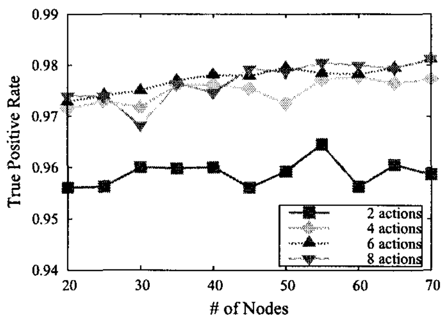
The decision maker refers to using accumulated output from the neural network to determine whether the corresponding user-input data is likely from a bot or a human player. Different algorithms can be applied to consolidate accumulated classifications. We employ a simple "voting" scheme: if the majority of the neural network output classifies the user-input actions as those of a bot, the decision will be that the game is operated by a bot, and vice versa. The decision process is a summary of the classifications of user-input actions over a period of time. While individual classification cannot be 100% correct, the more accumulated output, the more confidence we have in the decision. On the other hand, the more accumulated output, the more user-input actions are required, which translates to more data storage and longer time for decision making.

5.3.3 Performance Impact and Scalability

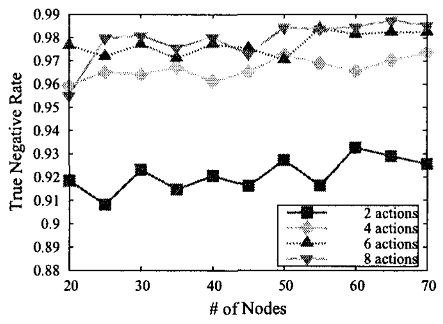
The nature of MMOGs dictates our design of the HOP system to be scalable and light-weight, limiting performance impacts on game clients and the server. At the client side, the system resource consumed by the collection of user-input actions is minor. In addition to the system resource of a game client, an MMOG player's gaming experience also depends on network performance. Since the user-input actions are short messages, 16 bytes of data per user-input action, the additional bandwidth consumption induced by the client-side exporter is negligible. The presence of the exporter thus is imperceptible for end users. At the server side, the scalability is critical to the success of our HOP system. The server-side analyzer is very efficient in terms of memory and CPU usage, which is shown in Section 5.4.4. The size of additional memory consumed per player is comparable to the size of the player's avatar name. A single processor core is capable of processing tens of thousands of users simultaneously in real-time. Therefore, the HOP system is scalable to the heavy workload at a game server.

5.4 Experiments

In this section, we evaluate the efficacy of our HOP system through a series of experiments, in terms of detection accuracy, detection speed, and system overhead. The metrics we use for detection accuracy include true positive rate and true negative rate. The true positive rate is the percentage of bots that are correctly identified, while the true negative rate is the percentage of humans that are correctly identified. The de-



(a) True Positive Rate



(b) True Negative Rate

Figure 5.9: True Positive and Negative Rates vs. # of Accumulated Actions and # of Nodes

tection speed is determined by the total number of actions needed to make decisions and the average time cost per action. In general, the larger the number of actions required for decisions and the higher the average time cost per action, the slower the detection speed becomes.

Table 5.2: True Positive and Negative Rates vs. Thresholds and # of Accumulated Outputs

Threshold	# of Accumulated Outputs										
	1	3	5	7	9	11	13	15	17	19	21
0.25	0.978 0.959	0.995 0.977	0.997 0.983	0.999 0.987	0.999 0.990	1.000 0.992	1.000 0.994	1.000 0.994	1.000 0.996	1.000 0.996	1.000 0.997
0.5	0.961 0.976	0.991 0.990	0.997 0.994	0.998 0.996	0.999 0.998	1.000 0.998	1.000 0.998	1.000 0.998	1.000 0.999	1.000 0.998	1.000 0.999
0.75	0.926 0.985	0.980 0.995	0.992 0.997	0.997 0.998	0.998 1.000	0.998 0.999	0.998 0.999	1.000 1.000	1.000 1.000	1.000 0.999	1.000 1.000
0.9	0.859 0.991	0.935 0.998	0.964 0.999	0.980 0.999	0.985 1.000	0.996 0.999	0.995 0.999	0.996 1.000	0.995 1.000	0.998 0.999	0.998 1.000
0.95	0.775 0.994	0.856 0.999	0.895 0.999	0.922 0.999	0.940 1.000	0.947 0.999	0.958 0.999	0.969 1.000	0.976 1.000	0.975 0.999	0.983 1.000
0.975	0.624 0.996	0.668 0.999	0.700 0.999	0.723 0.999	0.737 1.000	0.757 0.999	0.770 0.999	0.776 1.000	0.792 1.000	0.796 0.999	0.804 1.000

5.4.1 Experimental Setup

Our experiments are based on 95 hours of traces, including 55 hours of human traces and 40 hours of game bot traces. In total, these traces contain 3,000,066 raw user-input events and 286,626 user-input actions, with 10 bot instances and 30 humans involved. The 10 bot instances are generated by running the Glider bot with 10 different profiles. The human players are a diverse group, including men and women with different ages and different levels of gaming experience. The more detailed trace information has been given in Section 5.2.2.

The experiments are conducted using 10-fold cross validation. Each test is performed on a different human or bot that is left out of the training set for that test. Therefore, to validate a given configuration, 20 different partitions are created, one for each of the 10 bots and 10 sets of three humans. The partitions consist of a training set of either 9 bots and 30 humans or 10 bots and 27 humans, and a test set of either one bot or three humans. Thus, each test is performed on unknown data that the system has not yet been trained on.

5.4.2 Detection Results

The HOP system has four configurable parameters: the number of actions per block, the number of nodes, the threshold, and the number of outputs per output block. The first two parameters mainly determine the size and complexity of the neural network, while the second two parameters largely affect the detection performance of the entire system. The threshold determines how a neural network output is interpreted: a value over the threshold indicates a bot, while a value under the threshold indicates a human. Note that humans have a value of 0.0 and bots have a value of 1.0 in the training of the neural network.

We first configure the number of actions per block and the number of nodes. The true positive and true negative rates with different numbers of actions and different numbers of nodes are shown in Figure 5.9 (a) and (b), respectively. These tests are performed with a default threshold of 0.5. The neural network becomes more accurate as more actions are provided, but we see diminishing returns in accuracy as the number of actions increases, e.g., going from 4 actions to 6 actions requires 50% more input but only provides a relatively small increase in the overall accuracy.

In most cases, the binomial theorem predicts that combining three decisions for the 4-action neural network should be more accurate than combining two decisions for the 6- or 8-action neural networks. Therefore, we choose to use a neural network with 4 actions as input, which gives true positive and negative rates of 0.971-0.977 and 0.959-0.973, respectively.

The overall true positive and negative rates do not always grow as the number of nodes increases. At some points, increasing the number of nodes no longer improves the true positive or negative rates and the neural network starts to over-fit the training set. A neural network of 40 nodes provides a true positive rate of 0.976 and a true negative rate of 0.961, which is the best combination of true positive and true negative rates with 4 actions as input. Therefore, we set up the neural network based on this configuration.

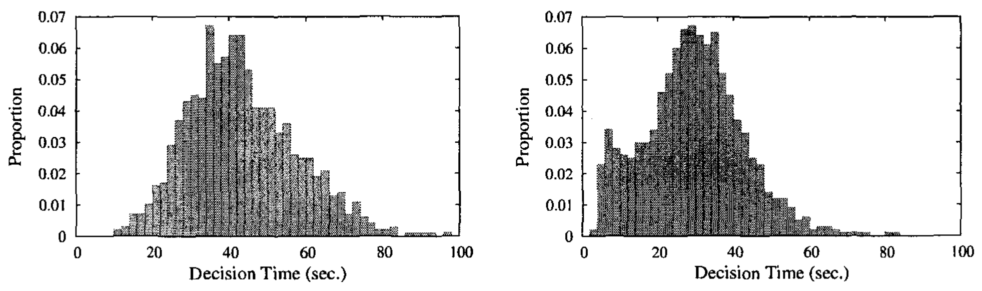
With the neural network configured, the threshold and the number of outputs per block determine the overall performance of the system. The threshold can be increased or decreased from the default value of 0.5 to bias the neural network towards bots or humans, improving the true positive rate or the true negative rate, respectively. The number of outputs per block affects both the detection accuracy and the detection speed of the system. As the number of outputs per block increases, the detection accuracy of the system increases, but the detection speed decreases as more neural network outputs are needed to make decisions.

The true positive and negative rates with different thresholds and different numbers of outputs for bots and humans are listed in Table 5.2. The top number in each cell is the true positive rate and the bottom number is the true negative rate. The neural network has 40 nodes and takes 4 actions as input. There are a number of settings that allow for a true positive or true negative rate of 1.0, though not both. To avoid a false positive—mistaking a human for a bot, we prefer a high true negative rate. The smallest number of outputs per block that achieves a true negative rate of 1.0 is 9 outputs per block with the threshold of 0.75, which gives a true positive rate of 0.998.

Table 5.3: True Positive Rates for Bots

Bots									
#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
0.988	1.000	0.998	1.000	1.000	1.000	1.000	1.000	0.998	1.000

With the fully configured system (40 nodes, 4-action input, the threshold of 0.75,



(a) Bot (b) Human

Figure 5.10: Decision Time Distribution

and 9 outputs per block), Table 5.3 lists the true positive and negative rates for each of the individual bots in our traces. The true negative rates are 1.0 for all of the humans, so none of the human players in our traces are misclassified as bots. The true positive rates are between 0.988 and 1.000 for the bots in our traces, with the average true positive rate of 0.998.

The detection speed of the system is a function of the total number of actions required for decision making and the average time cost per action. The total number of actions is 36 (i.e., 9 outputs $\times$ 4 actions per output). The time cost per action varies. The average time cost per action, ignoring idle periods longer than 10 seconds, is 1.10 seconds. If a player is idle, strictly speaking, no one is "operating" the game, so no decision can be made. Of course, idle players (bots or humans) are not performing any actions and should not be a concern. Based on the total number of actions and the average time cost per action, Figure 5.10 illustrates the decision time distribution for bots and humans. From the decision time distribution, we can see that our HOP system is able to make decisions for capturing bots within 39.60 seconds on average.

Note that we perform the same experiments with BP neural networks and observe that the cascade neural network is more accurate in bot classification than BP neural networks that use incremental, quick propagation, or resilient propagation method.

5.4.3 Detection of Other Game Bots

To further test our HOP system, without retraining the neural network, we perform a smaller experiment on a different game bot from a different game. While Diablo 2 is not an MMOG, it has an MMOG-like economy (items may be traded with thousands of other players) and is also plagued by game bots. This set of experiments studies MMBot, a popular free bot for Diablo 2 that is built using the AutoIt scripting language [10]. Similar to Glider, MMBot automates various tasks in the game to accumulate treasure or experience. However, unlike Glider, MMBot does not read the memory space of the game, but instead is based entirely on keyboard/mouse automation, and pixel scanning. As Diablo 2 has a much different interface (top-down isometric view rather than first person view like World of Warcraft) and much different controls, the purpose of these experiments is to test how general our system is and to show that it is not limited to any specific bot or game.

We collect a total of 20 hours of Diablo 2 traces, both bot and human. We run MMBot for 10 hours and have 5 humans play Diablo 2 for a total of 10 hours. We then reuse our existing neural network (40 nodes, 4 action-input, 9 inputs per block) with the adjusted threshold value to optimize our detection results. Without retraining, the neural network achieves a true positive rate of 0.864 on the bot and a true negative rate of 1.0 on the human players. This result shows that our HOP system is able to capture certain invariants in the behavior of bots across different games and different bot implementations, indicating the possible potential of HOP-based systems for other applications.

5.4.4 System Overhead

Our proposed system at the server side (i.e., the server-side analyzer) is required to process thousands of users simultaneously in real-time, so it must be efficient in terms of memory and computation. Now we estimate the overhead of the analyzer for supporting 5,000 users, far more than the regular workload of a typical World of Warcraft server. The analyzer process, which includes the neural network, is profiled using `valgrind` and consumes only 37 KBytes of memory during operation. The prototype of our system is designed to use a single-thread multiple-client model with time-multiplexing, and thus only one process is used. Of course, additional processes could be used to process in parallel.

The primary memory requirement is to accommodate the accumulated user-input actions and neural network outputs for each online user. A single user-input action consumes 16 bytes, 4 bytes each for distance, duration, and displacement, and 2 bytes each for virtual key and angle. A block of 4 user-input actions consumes 64 bytes. A block of up to 16 neural network outputs requires 2 bytes as a bit-array. The per-user memory requirement is approximately 66 bytes, barely more than the maximum length of account names on World of Warcraft, which is 64 bytes. If 66 bytes is scaled to 5,000 online users, this is only 330 KBytes in total, which is negligible considering that the game currently stores the position, level, health, and literally dozens of other attributes and items for each user.

The computational overhead is also very low. The computation time for processing all 95 hours of traces is measured using the Linux `time` command. The analyzer can process the full set of traces, over 286,626 user-input actions, in only 385 milliseconds on a Pentium 4 Xeon 3.0Ghz. In other words, the analyzer can process approximately 296 hours of traces per second using a single CPU. A server with 5,000 users would generate approximately 1.38 hours of traces per second, a tiny fraction of the above processing rate.

5.5 Related Work

Exploiting online games has attracted increasing interest in recent years. Yan *et al.* [133] summarized commonly-used exploiting methods in online games and categorized them along three dimensions: vulnerability, consequence, and exploiter. In addition, they pointed out that fairness should be taken into account to understand game exploits. Webb *et al.* [126] presented a different classification of game exploits. They categorized 15 types of exploits into four levels: game, application, protocol, and infrastructure, and discussed countermeasures for both client-server and peer-to-peer architectures. Muttik [89] surveyed security threats emerging in MMOGs, and discussed potential solutions to secure online games from multiple perspectives including technology, economy, and human factor. Hoglund and McGraw [54] provided a comprehensive coverage of game exploits in MMOGs, shedding light on a number of topics and issues.

5.5.1 Anti-Cheating

With the ever-increasing severity of game exploits, securing online games has received wide attention. The research work on anti-cheating generally can be classified into two categories: game cheating prevention and game cheating detection. The former refers to the mechanisms that deter game cheating from happening and the latter comprises the methods that identify occurrences of cheating in a game. For MMOGs, a cheat-proof design, especially the design of the game client program and the communication protocol, is essential to prevent most of game exploits from occurring. This is because (1) the client program of an MMOG is under the full control of a game player and (2) the communication at the client side might be manipulated for the advantage of player.

The prevention of game exploits has been the subject of a number of works. Baughman *et al.* [8] uncovered the possibility of time cheats (e.g., look-ahead and suppress-correct cheats) through exploiting communication protocols for both centralized and distributed online games, and designed a lockstep protocol, which tightly synchronizes the message communication via two-phase commitment, to prevent cheats. Following their work, a number of other time-cheat-resistant protocols [29, 34, 23] have been developed. In [86], Mönch *et al.* proposed a framework for preventing game client programs from being tampered with. The framework employs mobile guards, small pieces of code dynamically downloaded from the game server, to validate and protect the game client. Yampolskiy *et al.* [132] devised a protection mechanism for online card games, which embeds CAPTCHA tests in the cards by replacing the card face with text. Besides software approaches, hardware-based approaches to countering game exploits have also been proposed. Golle *et al.* [50] presented a special hardware device that implements physical CAPTCHA tests. The device can prevent game bots based on the premise that physical CAPTCHA tests such as pressing certain buttons are too difficult for bots to resolve without human involvement.

In practice, it is extremely hard to eliminate all potential game exploits. Thus, accurate and quick detection of game exploits is critical for securing on-line games. Since game bots are a commonly-used exploit, a fair amount of research has focused on detecting and countering them. Based on traffic analysis, Chen *et al.* [24] found that the traffic generated by the official client differs from that generated by standalone bot programs. Their approach, however, is not effective against recent game bots, as the majority of current MMOG bots interact with official clients. In [25, 26], the difference of movement paths between human players and bots in a first-person shooter (FPS) game is revealed and then used for the development of trajectory-based detection methods. However, it is unlikely that this type of detection method can achieve similar speed and accuracy in MMOGs, because maps used in MMOGs are much larger than those in FPS games and avatar trajectories in MMOGs are far more complicated. Indeed, Mitterhofer *et al.* [83] used movement paths in World of Warcraft and their method requires from 12 to 60 minutes to detect game bots. Thawonmas *et al.* [115] introduced a behavior-based bot detection method, which relies on discrepancies in action frequencies between human players and bots. However, compared to our work, the metric used for their detection, action frequency, is coarse-grained and has low discriminability, resulting in low detection ratio (0.36 recall ratio on average) and long detection time (at least 15 minutes).

As game clients in general cannot be trusted, usually the detection decision is made at servers. Schluessler *et al.* [106] presented a client-side detection scheme, which detects input data generated by game bots by utilizing special hardware. The hardware is used to provide a tamper-resistant environment for the detection module. The detection module compares the input data generated by input devices (mouse and keyboard) with those consumed by the game application and fires an alert once a discrepancy is found.

5.5.2 Behavioral Biometrics

The idea of HOPs is largely inspired by behavioral biometrics based on keystroke dynamics [60, 87, 11, 96] and mouse dynamics [2, 43, 103]. Analogous to handwritten signatures, keystroke dynamics and mouse dynamics are regarded as unique to each person. Therefore, their applications in user authentication and identification have been extensively investigated [60, 87, 11, 96, 103, 43, 2]. Generating synthetic mouse dynamics from real mouse actions has also been studied [90, 91]. In spite of the fact that our system also utilizes the characteristics of keystroke and mouse dynamics, it significantly differs from aforementioned biometric systems in that our system leverages the distinction on game play between human players and game bots, which is reflected by keystroke and mouse dynamics, to distinguish human players from game bots. In contrast, those biometric systems exploit the uniqueness of keystroke dynamics or mouse dynamics for identification, i.e., matching a person with his/her identity on the basis of either dynamics.

5.6 Conclusion

In this chapter, we presented a game bot defense system that utilizes HOPs to detect game bots. The proposed HOPs leverage the differences of game playing behaviors such as keyboard and mouse actions between human players and game bots to identify bot programs. Compared to conventional HIPs such as CAPTCHAs, HOPs are transparent to users and work in a continuous manner. We collected 95-hour user-input traces from World of Warcraft. By carefully analyzing the traces, we revealed that there exist significant differences between bots and humans in a variety of characteristics derived from game playing actions, which motivate the design of the proposed HOP defense system.

The HOP defense system comprises a client-side exporter and a server-side analyzer. The exporter is used to transmit a stream of user-input actions and the analyzer is used to process the action stream to capture bots. The core of the analyzer is a cascade-correlation neural network, which takes an action stream as input and determines if the stream generator is a bot or a human player. We also employed a simple voting algorithm to further improve detection accuracy. Based on the collected user-input traces, we conducted a series of experiments to evaluate the effectiveness of the defense system under different configurations. Our results show that the system can detect over 99% of current game bots with no false positives within a minute and the overhead of the detection is negligible or minor in terms of induced network traffic, CPU, and memory cost. As our detection engine only relies on user-input information, our HOP system is generic to MMOGs.

Chapter 6

Conclusions and Future Work

This dissertation explores applications of information theory and statistical learning to anomaly detection. Specifically, we address two very important and challenging problems in network and system security, (1) detecting covert timing channels, and (2) determining if a user is a human or a bot. For the first problem, we developed an entropy-based approach for detecting covert timing channels. For the second problem, we developed a hybrid classification system, based on entropy and statistical learning, for detecting chat bots, and a game bot defense system, based on statistical learning, for detecting game bots.

To detect covert timing channels, which could leak sensitive information from a system or network, we first studied covert timing channel design, and modeled, simulated, and tested different covert timing channels to better understand their characteristics. Based on the observation that the creation of a covert timing channel has certain effects on the entropy of the original process, we developed an entropy-based approach to detecting covert timing channels. We implemented our entropy-based approach using entropy and corrected conditional entropy. Our experimental results show that our entropy-based approach is sensitive to current covert timing channels and capable of detecting them.

To defend against bots, which abuse various network applications, including chat, online games, web sites, and so on, we developed two different detection systems for bots in different network applications, chat and online games.

First, we conducted a large-scale measurement study on a major commercial chat service, Yahoo! Chat, capturing over 1400 hours of chat logs. Based on the measurement study, we proposed a hybrid classification system to differentiate bots from humans. The hybrid classification system consists of entropy-based (entropy and corrected conditional entropy) and statistical-learning-based (Bayesian classification) classifiers. The two classifiers complement each other in detection. The entropy-based classifier is more effective against unknown or zero-day chat bots, whereas the statistical-learning-based classifier is faster against known chat bots. Our experimental results show that the hybrid classification system is able to quickly classify known chat bots and accurately classify previously unknown chat bots.

Second, we collect game play traces for a popular massive multiplayer online game, World of Warcraft, capturing 95 hours of game play. The traces show various differences between humans and bots in their user-input characteristics. In addition, we observe that some actions with the game are difficult for bots to perform in a human-like manner due to the need to process complex visuals in real-time. Based on these observations, we developed a HOP-based game bot defense system that analyzes user-input actions and uses a cascade-correlation neural network to determine if users are humans or bots. Our experimental results show that the HOP-based game bot defense system is highly effective against current game bots, raising the bar for attackers.

6.1 Future Work

Our future work will pursue several extensions to our entropy-based approach to detecting covert channels, our hybrid approach to detecting chat bots, and our statistical learning and human-observational-proof-based approach to detecting online game bots.

First, we will determine how our entropy-based approach and other detection methods limit the capacity of covert timing channels. By measuring how detection methods constrain capacity, we get a broader view of how detection methods enhance overall covert timing channel defense. In addition, we believe that this further exploration will lead to better detection methods that further limit covert timing channel capacities.

Second, we also will look at more advanced and human-like chat bots that could evade our hybrid classification system. It is possible for more advanced chat bots to look more human-like in their inter-message delay and message size statistics. By combining the characteristics of responder bots and replay bots, a new bot could take on the approximate inter-message delay and message size entropy of humans and possibly evade our hybrid classification system.

Third, we also plan to extend our HOP-based detection system to other interactive applications outside of online games. A number of interactive web applications, such as social networking sites, blogs, and web-based e-mail services, all of which are plagued by bots, could represent possible applications for such a system.

Appendix A

Chat Bot Examples

A.0.1 Response Example

```
bot:  user1, that's a damn good question.

bot:  user1, To know more about Seventh-day Adventist; visit http://www.sda.org

bot:  user2, no!  don't leave me.

bot:  user1, too much coffee tonight?

bot:  user2, boy, you're just full of questions, aren't you?

bot:  user2, lots of evidence for evolution can be found here http://www.talkorigins.org/faqs
```

In the above example, the bot uses a template with three parts to post links:

[username], [link description phrase] [link].

A.0.2 Synonym Example

```
bot:  Allo Hunks!  Enjoy Marjorie!  Check My Free Pics
```

```
bot:  What's happening Guys!  Marjorie Here!  See more of me at My Free Pics

bot:  Hi Babes!  I am Marjorie!  Rate My Live Cam

bot:  Horny lover Guys!  Marjorie at your service!  Inspect My Site

bot:  Mmmm Folks!  Im Marjorie!  View My Webpage
```

A.0.3 Padding Example

```
bot:  anyone boredjn wanna chat?uklcss

bot:  any guystfrom the US/Canada hereiqjss

bot:  hiyafxqss

bot:  ne1 hereqbored?fiqss

bot:  ne guysmwanna chat?  ciuneed some1 to make megsmile :-)pktpss
```

Bibliography

[1] JOHAN AGAT. Transforming out timing leaks. In *Proceedings of the 2000 SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, January 2000.

[2] AHMED AWAD E. AHMED AND ISSA TRAORE. A new biometric technology based on mouse dynamics. *IEEE Trans. on Dependable and Secure Computing (TDSC)*, 4(3), 2007.

[3] L. VON AHN, M. BLUM, N. HOPPER, AND J. LANGFORD. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, Warsaw, Poland, May 2003.

[4] PAUL BÄCHER, THORSTEN HOLZ, MARKUS KÖTTER, AND GEORG WICHERSKI. Know your enemy: Tracking botnets, 2005. `http://www.honeynet.org/papers/bots` [Accessed: Jan. 25, 2008].

[5] SARAH BACON. Chat rooms follow-up. `http://www.ymessengerblog.com/blog/2007/08/21/chat-rooms-follow-up/` [Accessed: Jan. 25, 2008].

[6] SARAH BACON. Chat rooms update. `http://www.ymessengerblog.com/blog/2007/08/24/chat-rooms-update-2/` [Accessed: Jan. 25, 2008].

[7] SARAH BACON. New entry process for chat rooms. `http://www.ymessengerblog.com/blog/2007/08/29/new-entry-process-for-chat-rooms/` [Accessed: Jan. 25, 2008].

[8] NATHANIEL E. BAUGHMAN AND BRIAN NEIL LEVINE. Cheat-proof playout for centralized and distributed online games. In *Proceedings of the 20th IEEE INFOCOM*, Anchorage, AK, USA, April 2001.

[9] BBC NEWS STAFF. Legal battle over warcraft 'bot'. `http://news.bbc.co.uk/2/hi/technology/7314353.stm` [Accessed: Jan. 30, 2009].

[10] JONATHAN BENNETT. AutoIt: Automate and script windows tasks. `http://www.autoit.com/` [Accessed: Apr. 20, 2009].

[11] FRANCESCO BERGADANO, DANIELE GUNETTI, AND CLAUDIA PICARDI. User authentication through keystroke dynamics. *ACM Trans. on Information and System Security (TISSEC)*, 5(4), 2002.

[12] VINCENT BERK, ANNARITA GIANI, AND GEORGE CYBENKO. Covert channel detection using process query systems. In *Proceedings of FLOCON 2005*, September 2005.

[13] VINCENT BERK, ANNARITA GIANI, AND GEORGE CYBENKO. Detection of covert channel encoding in network packet delays. Technical Report TR2005-536, Dartmouth College, Computer Science, Hanover, NH., USA, August 2005.

[14] R. E. BLAHUT. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, 18(4), July 1972.

[15] BLIZZARD ENTERTAINMENT. MDY industries, LLC., vs. Blizzard Entertainment, Inc., and Vivendi Games, Inc. `http://gamepolitics.com/images/legal/blizz-v-MDY.pdf` [Accessed: Jan. 30, 2009].

[16] BLIZZARD ENTERTAINMENT. World of Warcraft subscriber base reaches 11.5 million worldwide. `http://eu.blizzard.com/en/press/081223.html` [Accessed: Jul. 24, 2009].

[17] JEREMY BLOSSER AND DAVID JOSEPHSEN. Scalable centralized bayesian spam mitigation with bogofilter. In *Proceedings of the 2004 USENIX Systems Administration Conference (LISA'04)*, Atlanta, GA., USA, November 2004.

[18] SERDAR CABUK. *Network Covert Channels: Design, Analysis, Detection, and Elimination.* PhD thesis, Purdue University, West Lafayette, IN., USA, December 2006.

[19] SERDAR CABUK, CARLA BRODLEY, AND CLAY SHIELDS. IP covert timing channels: Design and detection. In *Proceedings of the 2004 ACM Conference on Computer and Communications Security*, October 2004.

[20] CHRISTIAN CACHIN. An information-theoretic model for steganography. *Information and Computation*, 192(1), 2004.

[21] PATRICK CALDWELL. Blizzard bans 59,000 WOW accounts. `http://www.gamespot.com/news/6154708.html` [Accessed: Aug. 13, 2009].

[22] JIN CAO, WILLIAM S. CLEVELAND, DONG LIN, AND DON X. SUN. On the nonstationarity of internet traffic. In *Proceedings of the 2001 ACM SIGMETRICS/Performance*, June 2001.

[23] BEI DI CHEN AND MUTHUCUMARU MAHESWARAN. A cheat controlled protocol for centralized online multiplayer games. In *Proceedings of the 3rd ACM SIGCOMM NetGames*, Portland, OR, USA, August 2004.

[24] KUAN-TA CHEN, JHIH-WEI JIANG, POLLY HUANG, HAO-HUA CHU, CHIN-LAUNG LEI, AND WEN-CHIN CHEN. Identifying MMORPG bots: A traffic analysis approach. In *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE'06)*, June 2006.

[25] KUAN-TA CHEN, ANDREW LIAO, HSING-KUO KENNETH PAO, AND HAO-HUA CHU. Game bot detection based on avatar trajectory. In *Proceedings of the 7th International Conference on Entertainment Computing*, Pittsburgh, PA, USA, September 2008.

[26] KUAN-TA CHEN, HSING-KUO KENNETH PAO, AND HONG-CHUNG CHANG. Game bot identification based on manifold learning. In *Proceedings of the 7th ACM SIGCOMM NetGames*, Worcester, MA, USA, October 2008.

[27] THOMAS M. COVER AND JOY A. THOMAS. *Elements of information theory*. Wiley-Interscience, New York, NY., USA, 1991.

[28] DAN CRISLIP. Will Blizzard's spam-stopper really work? `http://www.wowinsider.com/2007/05/16/will-blizzards-spam-stopper-really-work/` [Accessed: Dec. 25, 2007].

[29] ERIC CRONIN, BURTON FILSTRUP, AND SUGIH JAMIN. Cheat-proofing dead reckoned multiplayer games (extended abstract). In *Proceedings of the 2nd International Conference on Application and Development of Computer Games*, Hong Kong, China, January 2003.

[30] DAVID DAGON, GUOFEI GU, CHRISTOPHER P. LEE, AND WENKE LEE. A taxonomy of botnet structures. In *Proceedings of the 2007 Annual Computer Security Applications Conference (ACSAC'07)*, Miami, FL., USA, December 2007.

[31] U. S. DEPARTMENT OF DEFENSE. Trusted computer system evaluation criteria, 1985.

[32] C. DEWES, A. WICHMANN, AND A. FELDMANN. An analysis of Internet chat systems. In *Proceedings of the 2003 ACM/SIGCOMM Internet Measurement Conference (IMC'03)*, Miami, FL., USA, October 2003.

[33] DIABLO 2 GUIDE. D2 bots. `http://www.diablo2guide.com/bots.php` [Accessed: Nov. 2, 2008].

[34] CHRIS GAUTHIER DICKEY, DANIEL ZAPPALA, VIRGINIA LO, AND JAMES MARR. Low latency and cheat-proof event ordering for peer-to-peer games. In *Proceedings of the 14th ACM NOSSDAV*, Cork, Ireland, June 2004.

[35] DMW WORLD. DMW anti-cheat system. `http://www.dmwworld.com/viewfaq/show/374` [Accessed: Aug. 13, 2009].

[36] R. DUDA, P. HART, AND D. STORK. *Pattern Classification*. Wiley-Interscience, New York, NY., USA, 2001.

[37] PETER EISLER. Reported raids on federal computer data soar, 2009. `http://www.usatoday.com/news/washington/2009-02-16-cyber-attacks_N.htm` [Accessed: Apr. 15, 2009].

[38] Adel El-Atawy and Ehab Al-Shaer. Building covert channels over the packet reordering phenomenon. In *Proceedings of the 2009 IEEE Conference on Computer Communications*, Rio de Janeiro, Brazil, April 2009.

[39] Even Balance Inc. PunkBuster online countermeasures. `http://www.evenbalance.com` [Accessed: Jul. 9, 2008].

[40] Exploits R Us. Ultima Online bots and cheats. `http://www.exploitsrus.com/uo/bots.html` [Accessed: Nov. 2, 2008].

[41] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, 1990.

[42] Gina Fisk, Mike Fisk, Christos Papadopoulos, and Josh Neil. Eliminating steganography in internet traffic with active wardens. In *Proceedings of the 2002 International Workshop on Information Hiding*, October 2002.

[43] Hugo Gamboa and Ana Fred. A behavioral biometric system based on human computer interaction. In *Proceedings of SPIE: Biometric Technology for Human Identification*, volume 5404, 2004.

[44] Steven Gianvecchio and Haining Wang. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security (CCS'07)*, Alexandria, VA., USA, October 2007.

[45] Steven Gianvecchio, Haining Wang, Duminda Wikesekera, and Sushil Jajodia. Model-based covert timing channels: Automated modeling and evasion. In *Proceedings of the 2008 Symposium on Recent Advances in Intrusion Detection*, September 2008.

[46] Steven Gianvecchio, Mengjun Xie, Zhenyu Wu, and Haining Wang. Measurement and classification of humans and bots in internet chat. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, USA, July 2008.

[47] John Giffin, Rachel Greenstadt, Peter Litwack, and Richard Tibbetts. Covert messaging through TCP timestamps. In *Proceedings of the 2002 International Workshop on Privacy Enhancing Technologies*, April 2002.

[48] James Giles and Bruce Hajek. An information-theoretic and game-theoretic study of timing channels. *IEEE Transactions on Information Theory*, 48(9), September 2002.

[49] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *Proceedings of the USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, Cambridge, MA., USA, April 2007.

[50] PHILIPPE GOLLE AND NICOLAS DUCHENEAUT. Preventing bots from playing online games. *Computers in Entertainment*, 3(3), 2005.

[51] PAUL GRAHAM. A plan for spam, 2002. `http://www.paulgraham.com/spam.html` [Accessed: Jan. 25, 2008].

[52] GUOFEI GU, PHILLIP PORRAS, VINOD YEGNESWARAN, MARTIN FONG, AND WENKE LEE. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *Proceedings of the 2007 USENIX Security Symposium (Security'07)*, Boston, MA., USA, August 2007.

[53] GUOFEI GU, JUNJIE ZHANG, AND WENKE LEE. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 2008 Annual Network and Distributed System Security Symposium (NDSS'08)*, San Diego, CA., USA, February 2008.

[54] GREG HOGLUND AND GARY MCGRAW. *Exploiting Online Games: Cheating Massively Distributed Systems.* No Starch Press, 2007.

[55] AMIR HOUMANSADR, NEGAR KIYAVASH, AND NIKITA BORISOV. RAINBOW: A robust and invisible non-blind watermark for network flows. In *Proceedings of the 2009 ISOC Network and Distributed System Security Symposium*, February 2009.

[56] JIM HU. AOL: spam and chat don't mix. `http://www.news.com/AOL-Spam-and-chat-dont-mix/2100-1032_3-1024010.html` [Accessed: Jan. 7, 2008].

[57] JIM HU. Shutting of MSN chat rooms may open up IM. `http://www.news.com/Shutting-of-MSN-chat-rooms-may-open-up-IM/2100-1025_3-5082677.html` [Accessed: Jan. 7, 2008].

[58] WEI-MING HU. Reducing timing channels with fuzzy time. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, May 1991.

[59] RAYMOND B. JENNINGS III, ERICH M. NAHUM, DAVID P. OLSHEFSKI, DEBANJAN SAHA, ZON-YIN SHAE, AND CHRIS WATERS. A study of internet instant messaging and chat protocols. *IEEE Network*, Vol. 20(No. 4):16–21, 2006.

[60] RICK JOYCE AND GOPAL GUPTA. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2), 1990.

[61] M. H. KANG, I. S. MOSKOWITZ, AND D. C. LEE. A network version of the pump. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, May 1995.

[62] Myong H. Kang and Ira S. Moskowitz. A pump for rapid, reliable, secure communication. In *Proceedings of the 1993 ACM Conference on Computer and Communications Security*, November 1993.

[63] Myong H. Kang, Ira S. Moskowitz, and Stanley Chincheck. The pump: A decade of covert fun. In *Proceedings of the 2005 Annual Computer Security Applications Conference*, December 2005.

[64] Christoph Karlberger, Gnther Bayler, Christopher Kruegel, and Engin Kirda. Exploiting redundancy in natural language to penetrate bayesian spam filters. In *Proceedings of the USENIX Workshop on Offensive Technologies*, Boston, MA., USA, August 2007.

[65] Richard A. Kemmerer. A practical approach to identifying storage and timing channels. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, April 1982.

[66] Richard A. Kemmerer. A practical approach to identifying storage and timing channels: Twenty years later. In *Proceedings of the 2002 Annual Computer Security Applications Conference*, December 2002.

[67] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based Spyware Detection. In *Proceedings of the 15th USENIX Security Symposium*, Vancouver, Canada, August 2006.

[68] Brian Krebs. Yahoo! messenger network overrun by bots. `http://blog.washingtonpost.com/securityfix/2007/08/yahoo_messenger_network_overru.html` [Accessed: Dec. 18, 2007].

[69] Urmila Kukreja, William E. Stevenson, and Frank E. Ritter. RUI - recording user input from interfaces under Windows and Mac OS X. *Behavior Research Methods, Instruments, and Computers*, 38(4):656–659, 2006.

[70] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10), October 1973.

[71] L.M. Leemis and S. K. Park. *Discrete-Event Simulation: A First Course.* Prentice-Hall, Upper Saddle River, New Jersey, 2006.

[72] Kang Li and Zhenyu Zhong. Fast statistical spam filter by approximate classifications. In *Proceedings of 2006 ACM/SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, St. Malo, France, June 2006.

[73] Zhijun Liu, Weili Lin, Na Li, and D. Lee. Detecting and filtering instant messaging spam - a global and personalized approach. In *Proceedings of the IEEE Workshop on Secure Network Protocols (NPSEC'05)*, Boston, MA., USA, November 2005.

[74] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *Proceedings of the 2005 Conference on Email and Anti-Spam (CEAS'05)*, Mountain View, CA., USA, July 2005.

[75] Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. Cloak: A ten-fold way for reliable covert communications. In *Proceedings of 2009 European Symposium on Research in Computer Security*, September 2007.

[76] Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. TCP covert timing channels: Design and detection. In *Proceedings of 2008 Dependable Systems and Networks*, June 2008.

[77] Mohammad Mannan and Paul C. van Oorschot. On instant messaging worms, analysis and countermeasures. In *Proceedings of the ACM Workshop on Rapid Malcode*, Fairfax, VA., USA, November 2005.

[78] McAfee. Mcafee virtual criminology report: Cybercrime versus cyberlaw, 2008. `http://www.mcafee.com/us/research/criminology_report/virtual_criminology_report/` [Accessed: Apr. 15, 2009].

[79] Corynne McSherry. A new gaming feature? spyware. `http://www.eff.org/deeplinks/2005/10/new-gaming-feature-spyware` [Accessed: Jul. 9, 2008].

[80] MDY Industries. MMO glider. `http://www.mmoglider.com/` [Accessed: Nov. 2, 2008].

[81] Wanda Meloni. State of the game industry 2008. In *GameOn Finance Conference*, San Diego, CA, USA, October 2008.

[82] Elinor Mills. Yahoo! closes chat rooms over child sex concerns. `http://www.news.com/Yahoo-closes-chat-rooms-over-/child-sex-concerns/2100-1025_3-5759705.html` [Accessed: Jan. 27, 2008].

[83] Stefan Mitterhofer, Christian Platzer, Christopher Kruegel, and Engin Kirda. Server-side bot detection in massive multiplayer online games. *IEEE Security and Privacy*, 7(3), May/June 2009.

[84] Ashish Mohta. Bots are back in Yahoo! chat rooms. `http://www.technospot.net/blogs/bots-are-back-in-yahoo-chat-room/` [Accessed: Dec. 18, 2007].

[85] Ashish Mohta. Yahoo! chat adds CAPTCHA check to remove bots. `http://www.technospot.net/blogs/yahoo-chat-captcha-check-to-remove-bots/` [Accessed: Dec. 18, 2007].

[86] Christian Mönch, Gisle Grimen, and Roger Midtstraum. Protecting online games against cheating. In *Proceedings of the 5th ACM SIGCOMM NetGames*, Singapore, October 2006.

[87] FABIAN MONROSE AND AVIEL RUBIN. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM CCS*, Zurich, Switzerland, April 1997.

[88] IRA S. MOSKOWITZ AND MYONG H. KANG. Covert channels - here to stay? In *Proceedings of the 1994 Annual Conference on Computer Assurance*, June 1994.

[89] IGOR MUTTIK. Securing virtual worlds against real attacks. `http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_online_gaming.pdf` [Accessed: Nov. 2, 2008].

[90] AKIF NAZAR. Synthesis and simulation of mouse dynamics. Master's thesis, University of Victoria, October 2007.

[91] AKIF NAZAR, ISSA TRAORE, AND AHMED AWAD E. AHMED. Inverse biometrics for mouse dynamics. *International Journal of Pattern Recognition and Artificial Intelligence*, 22(3):461–495, 2008.

[92] PROKOFY NEVA. Bots back in the box. `http://www.secondlifeherald.com/slh/2006/11/bots_back_in_th.html` [Accessed: Nov. 2, 2008].

[93] TATERU NINO. Linden Lab taking action against landbots. `http://www.secondlifeinsider.com/2007/05/18/linden-lab-taking-action-against-landbots/` [Accessed: Jan. 7, 2008].

[94] NPROTECT. nProtect GameGuard. `http://eng.nprotect.com/nprotect_gameguard.htm` [Accessed: Jul. 9, 2008].

[95] MASSIVELY MULTIPLAYER OWNED. World of Warcraft bots and programs forum. `http://www.mmowned.com/forums/bots-programs/` [Accessed: Jul. 21, 2009].

[96] ALEN PEACOCK, XIAN KE, AND MATTHEW WILKERSON. Typing patterns: A key to user identification. *IEEE Security and Privacy*, 2(5), 2004.

[97] PAI PENG, PENG NING, AND DOUGLAS REEVES. On the secrecy of timing-based active watermarking trace-back techniques. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.

[98] PETITION ONLINE. Action against the Yahoo! bot problem petition. `http://www.petitiononline.com/` [Accessed: Dec. 18, 2007].

[99] PETITION ONLINE. AOL no more chat room spam petition. `http://www.petitiononline.com/` [Accessed: Dec. 18, 2007].

[100] PIROX. PiroX Bot - World of Warcraft bot. `http://www.piroxbots.com/` [Accessed: Jul. 25, 2009].

[101] A Porta, G Baselli, D Liberati, N Montano, C Cogliati, T Gnecchi-Ruscone, A Malliani, and S Cerutti. Measuring regularity by means of a corrected conditional entropy in sympathetic outflow. *Biological Cybernetics*, 78(1), January 1998.

[102] Mila D. Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. In *Proceedings of the 34th ACM POPL*, Nice, France, January 2007.

[103] Maja Pusara and Carla E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, Washington, DC, USA, October 2004.

[104] Rhabot. Rhabot - World of Warcraft bot. `http://www.rhabot.com/` [Accessed: Nov. 2, 2008].

[105] Roman Rosipal. *Kernel-Based Regression and Objective Nonlinear Measures to Assess Brain Functioning.* PhD thesis, University of Paisley, Paisley, Scotland, UK, September 2001.

[106] Travis Schluessler, Stephen Goglin, and Erik Johnson. Is a bot at the controls?: Detecting input data attacks. In *Proceedings of the 6th ACM SIGCOMM NetGames*, Melbourne, Australia, September 2007.

[107] Mike Schramm. Chat spam measures shut down multi-line reporting add-ons. `http://www.wowinsider.com/2007/10/25/chat-spam-measures-shut-down-multi-line-reporting-addons/` [Accessed: Jan. 17, 2008].

[108] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, Vol. 34(No. 1):1–47, 2002.

[109] Sarah H. Sellke, Chih-Chun Wang, and Saurabh Bagchi. TCP/IP timing channels: Theory to implementation. In *Proceedings of the 2009 IEEE Conference on Computer Communications*, Rio de Janeiro, Brazil, April 2009.

[110] Gaurav Shah, Andres Molina, and Matt Blaze. Keyboards and covert channels. In *Proceedings of the 2006 USENIX Security Symposium*, July–August 2006.

[111] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, July and October 1948.

[112] Chet Simpson. Yahoo! chat anti-spam resource center. `http://www.chatspam.org/` [Accessed: Sep. 25, 2007].

[113] Slashdot. Confessions of an Ultima Online gold farmer. `http://slashdot.org/games/05/01/26/1531210.shtml` [Accessed: Jul. 9, 2008].

[114] Symantec Security Response. W32.Imaut.AS worm. http://www.symantec.com/security_response/writeup.jsp?docid=2007-080114-2713-99 [Accessed: Jan. 25, 2008].

[115] Ruck Thawonmas, Yoshitaka Kashifuji, and Kuan-Ta Chen. Detection of MMORPG bots based on behavior analysis. In *Proceedings of 5th ACM International Conference on Advances in Computer Entertainment Technology (ACE'08)*, Yokohama, Japan, December 2008.

[116] The ALICE Artificial Intelligence Foundation. ALICE(Artificial Linguistice Internet Computer Entity). http://www.alicebot.org/ [Accessed: Jan. 25, 2008].

[117] The MMO RPG Exchange. World of Warcraft exchange. http://themmorpgexchange.com/ [Accessed: Jul. 25, 2009].

[118] Alan M. Turing. Computing machinery and intelligence. *Mind*, Vol. 59:433–460, 1950.

[119] Uber-Geek.com. Yahoo! responder bot. http://www.uber-geek.com/bot.html [Accessed: Jan. 18, 2008].

[120] Valve Corporation. Valve anti-cheat system (VAC). https://support.steampowered.com/kb_article.php?p_faqid=370 [Accessed: Jul. 9, 2008].

[121] Haining Wang, Danlu Zhang, and Kang G. Shin. Detecting SYN flooding attacks. In *Proceedings of the 21st IEEE INFOCOM*, New York, NY, USA, June 2002.

[122] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. Tracking anonymous peer-to-peer VoIP calls on the internet. In *Proceedings of the 2005 ACM Conference on Computer and Communications Security*, November 2005.

[123] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. Network flow watermarking attack on low-latency anonymous communication systems. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Washington, DC, USA, May 2007.

[124] Xinyuan Wang and Douglas S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proceedings of the 2003 ACM Conference on Computer and Communications Security*, October 2003.

[125] Zhenghong Wang and Ruby Lee. Covert and side channels due to processor architecture. In *Proceedings of the 2006 Annual Computer Security Applications Conference*, December 2006.

[126] Steven Daniel Webb and Sieteng Soh. Cheating in networked computer games – a review. In *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts*, Perth, Australia, September 2007.

[127] Walter Willinger, Vern Paxson, and Murad S. Taqqu. Self-similarity and heavy tails: Structural modeling of network traffic. In *Statistical Techniques and Applications*, pages 27–53. Verlag, 1998.

[128] Gregory L. Wittel and S. Felix Wu. On attacking statistical spam filters. In *Proceedings of the 2004 Conference on Email and Anti-Spam (CEAS'04)*, Mountain View, CA., USA, July 2004.

[129] WoW Panda. ZoloFighter - World of Warcraft bot. `http://www.zolohouse.com/wow/wowFighter/` [Accessed: Jul. 25, 2009].

[130] Mengjun Xie, Zhenyu Wu, and Haining Wang. HoneyIM: Fast detection and suppression of instant messaging malware in enterprise-like networks. In *Proceedings of the 2007 Annual Computer Security Applications Conference (ACSAC'07)*, Miami Beach, FL, USA, December 2007.

[131] Yahelite.org. Yahelite chat client. `http://www.yahelite.org/` [Accessed: Jan. 8, 2008].

[132] Roman V. Yampolskiy and Venu Govindaraju. Embedded non-interactive continuous bot detection. *Computers in Entertainment*, 5(4), 2007.

[133] Jeff Yan and Brian Randell. A systematic classification of cheating in online games. In *Proceedings of the 4th ACM SIGCOMM NetGames*, Hawthorne, NY, USA, October 2005.

[134] YazakPro.com. Yazak pro chat client. `http://www.yazakpro.com/` [Accessed: Jan. 8, 2008].

[135] Bill Yerazunis. CRM114 - the controllable regex mutilator, 2003. `http://crm114.sourceforge.net` [Accessed: Jan. 25, 2008].

[136] Wei Yu, Xinwen Fu, Steve Graham, Dong Xuan, and Wei Zhao. Dsss-based flow marking technique for invisible traceback. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Washington, DC., USA, May 2007.

[137] Jonathan A. Zdziarski. *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification.* No Starch Press, 2005.

[138] Li Zheng, Liren Zhang, and Dong Xu. Characteristics of network delay and delay jitter and its effect on oice over IP (VoIP). In *Proc. of the 2001 IEEE International Conf. on Communications*, June 2001.

VITA

Steven Gianvecchio

Steven Gianvecchio received his B.S. in Computer Science from the State University of New York at Brockport in 2001 and his M.S. in Computer Science from the College of William and Mary in 2006. He started his Ph.D. in Computer Science at the College of William and Mary in 2006. His research interests include networks, distributed systems, network monitoring, intrusion detection, traffic modeling, and covert channels. His current research focuses on information-theoretic and statistical learning methods for addressing different problems in network and system security, such as detecting covert channels or determining if users are humans or bots.