12-2012

Network Traffic Analysis Using Stochastic Grammars

Chen Lu
Clemson University, lu4@g.clemson.edu

Network Traffic Analysis Using Stochastic Grammars

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Electrical Engineering

by
Chen Lu
December 2012

Accepted by:
Dr. Richard R. Brooks, Committee Chair
Dr. Harlan B. Russell
Dr. Adam Hoover
Dr. Robert Lund

Abstract

Network traffic analysis is widely used to infer information from Internet traffic. This is possible even if the traffic is encrypted. Previous work uses traffic characteristics, such as port numbers, packet sizes, and frequency, without looking for more subtle patterns in the network traffic. In this work, we use stochastic grammars, hidden Markov models (HMMs) and probabilistic context-free grammars (PCFGs), as pattern recognition tools for traffic analysis.

HMMs are widely used for pattern recognition and detection. We use a HMM inference approach. With inferred HMMs, we use confidence intervals (CI) to detect if a data sequence matches the HMM. To compare HMMs, we define a normalized Markov metric. A statistical test is used to determine model equivalence. Our metric systematically removes the least likely events from both HMMs until the remaining models are statistically equivalent. This defines the distance between models. We extend the use of HMMs to PCFGs, which have more expressive power. We estimate PCFG production probabilities from data. A statistical test is used for detection.

We present three applications of HMM and PCFG detection to network traffic analysis. First, we infer the presence of protocol tunneling through Tor (the onion router) anonymization network. The Markov metric quantifies the similarity of network traffic HMMs in Tor to identify the protocol. It also measures communication noise in Tor network.

We use HMMs to detect centralized botnet traffic. We infer HMMs from botnet traffic data and detect botnet infections. Experimental results show that HMMs can accurately detect Zeus botnet traffic.

To hide their locations better, newer botnets have P2P control structures. Hierarchical P2P botnets contain recursive and hierarchical patterns. We use PCFGs to detect P2P botnet traffic. Experimentation on real-world traffic data shows that PCFGs can accurately differentiate between P2P botnet traffic and normal Internet traffic.

Dedication

This thesis is dedicated to my family, especially my wife Wei Lou. You have always supported and encouraged me throughout this work.

Acknowledgments

First of all, I want to thank my advisor, Dr. Richard R. Brooks. This dissertation could not have been done without your insightful directions, helpful suggestions, incredible patience and a lot of valuable time. Your guidance and inspiration during the past three and half years helped me to gain a thorough understanding of not only the research itself, but also the way to conduct research. Thank you for your support and encouragement that enables me to reach this goal.

Second, I would like to thank Dr. Harlan Russell, Dr. Adam Hoover and Dr. Robert Lund for being my committee members. Without the knowledge and techniques that I learned in your classes, a lot of the work in this dissertation could not become a reality. I also want to thank Dr. Lund for helping me on the mathematical approach that I used in my research.

Third, without the help and cooperation of previous and current students in my research group, it would take me much more time and effort to understand research concepts and conduct experiments. The communications with them always inspire interesting and useful ideas. It is my pleasure to work with these talented and helpful students. I would also like to thank Dr. Wenke Lee of the Georgia Institute of Technology for providing me with P2P botnet traffic to verify my approach.

I also want to thank my family. No matter what happens, you are always there to give me endless support. Your understanding and support give me courage and wisdom to overcome any difficulties in the past and in the future.

Last but not the least, I would like to give my acknowledgment to Air Force Office of Scientific Research for providing the financial support for my work. This material is based upon work supported by, or in part by, the Air Force Office of Scientific Research contract/grant number FA9550-09-1-0173.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the development of networked and digital communications, more and more devices are connected to the Internet, including computers, tablets and smartphones. As network communications become ubiquitous, increasingly confidential, sensitive and important information is traveling through the Internet. Access to this information is valuable; particularly unauthorized access by attackers. Analysis of network traffic data is important for both attackers and defenders.

In this work, we use pattern recognition and detection tools to extract useful information from traffic data, especially when the content of the data is not available. We focus on two stochastic grammars as our tools: hidden Markov models (HMMs), which are probabilistic regular grammars, and probabilistic context-free grammars (PCFGs). Using approaches we developed for these two stochastic grammars, we are able to infer the existence of protocols tunneled through the network, measure noise in the traffic patterns and detect traffic generated by specific network processes. This allows us to detect which protocols are used, even if the tunneled channel has been encrypted. This approach does not require computationally expensive cryptanalysis.

1.1 Network Traffic Analysis

Network traffic is usually encrypted to ensure communications security. Previous detection approaches use reverse engineering and decryption to reveal the traffic contents [26] [86]. This requires intensive manual mathematical analysis or brute-force attacks on the traffic data. Considering that simple web browsing may generate thousands of network packets, and breaking advanced encryption algorithms requires hundreds of computers and weeks of time, cryptanalysis is practical only for a few nation states for limited data volumes.

Network traffic analysis has become an active research topic. Song et al. infer user passwords from inter-keystroke timings sent through SSH [83]. The approach in [98] differentiates between camouflaging worm[1] traffic and normal traffic using a spectrum-based scheme on the traffic data. In [88], flow characteristics, such as bandwidth, packet timing, and burst duration, were examined to find evidence of botnet command and control[2] activity. Traffic timing analysis can reconstruct end-to-end VOIP communication systems [53]. A passive DNS traffic analysis system for detecting and tracking malicious flux networks is in [67].

In this work, we perform *traffic timing analysis*. Timing data can be easily obtained from traffic data, no matter whether the traffic is encrypted or not. Inter-packet timings provide information about the temporal properties of unknown underlying processes [59]. For example, some network processes (e.g. email and cloud services) need to contact outside sources repeatedly, and their timing patterns may have periodic components, while other processes (e.g. web browsing and file transferring) may have low-latency communications. Inter-packet delays filter out constant network latencies, and they are preserved across router hops during network transmissions [92]. Link padding, which pads inter-packet timings into constant delays, is suggested to evade traffic timing analysis. However, it will affect the performance of network processes and the work in [35, 36] shows that it is still vulnerable to statistical analysis. Since automated network processes all have timing profiles and encryption can not remove timing signatures from malicious processes, timing analysis has a variety of applications, such as detecting botnet traffic [44], revealing VOIP systems [20, 53], learning user's password from SSH traffic timings [83], detecting languages typed through SSH [19, 18], and breaking Tor anonymity [28].

Previous work [20, 44, 59, 92] used correlation of timing data for analysis and detection. Unlike these works, we consider the communication patterns in inter-packet timings caused by underlying network processes. We apply widely used pattern recognition tools, such as hidden Markov models [70] and probabilistic context-free grammars [13, 64], for detecting patterns in traffic timing data. In performing this work, a number of innovations of HMMs and PCFGs were needed.

[1]The camouflaging worm can intelligently manipulate its scan traffic over time. Therefore, it camouflages propagation from worm detection systems based on analyzing the traffic generated by worms [98].

[2]A botnet is a network of comprised computers, which are controlled by command and control servers for malicious purposes. Later sections will talk about botnets in details.

1.2 Stochastic Grammars

In recent years, syntactic pattern recognition methods are widely used for pattern classification and detection. For example, hidden Markov models (HMMs) have success in gesture [61], speech [70] and language [69, 84] recognition. In these applications, model parameters are inferred from training data sets. The inferred models are then applied to new observation streams to determine if the observed data is a representation of the constructed model [70, 22].

Traditionally, the Baum-Welch algorithm is used to infer the structure and parameters of the HMM, given an initial model and a sequence of observations [70]. As a result, this algorithm requires the *a priori* structural knowledge of the Markov process that produced the outputs. In [78, 77, 80], an approach is developed that derives the HMM state structure and transition matrix from available data without knowing the *a priori* structure of the HMM. However, this algorithm needs a parameter. In our work, we use the extension in [76] to derive minimum entropy HMMs with no *a priori* information. Using confidence intervals (CIs) of HMMs [22], we can detect whether or not a sequence of data matches the model. Detailed description of this approach is in Chapter 3.

When several models are inferred from similar data sets, a metric is useful for determining if one model is simply a different representation of the same process, or a representation of a very different system. Therefore, we derived a metric space for comparing HMMs based on the system statistics, instead of the model structure [57]. We use a statistical test to determine if two models are equivalent within a given level of significance. In addition, we use data sampling insights to systematically remove least likely events from both HMMs until the remaining models are statistically equivalent. The largest probability of all removed events defines the distance between two models. Chapter 3 provides an in-depth description of this metric space.

Despite these successes, HMMs may have difficulty detecting more complex patterns in applications, such as language understanding, translation, and recursive pattern detection [17, 37, 56]. This is because HMMs are equivalent to probabilistic regular grammars, which the simplest class in Chomsky Hierarchy of grammars [64]. It can only represent patterns in regular expressions, and can not detect recursive patterns [64]. On the other hand, the next more expressive computation grammar, context-free grammars (CFGs), have been developed to recognize and detect recursive patterns. More detailed discussion about Chomsky Hierarchy and the extension of HMMs to CFGs is provided in Chapter 2.

Probabilistic (stochastic) CFGs (PCFGs) are CFGs with each production rule augmented by a probability[3]. PCFGs are usually used for real world pattern recognition problems, because they can accommodate noise and pattern distortion, incorporate not only syntactic but also statistical information in data, and distinguish more plausible results from less plausible ones [58, 72]. PCFGs have been successfully applied to natural language processing [37, 58, 72], pattern analysis in RNA and protein sequences [31, 89, 90], and network data modeling [39].

Therefore, we use PCFGs to detect recursive patterns that can not be detected by HMMs. We estimate PCFG production probabilities using maximum-likelihood (ML) method from data set [25, 24, 27]. Traditionally, the inside-outside algorithm is used for data classification purposes [51, 52]. It finds the most likely PCFG that generates the data set, from several candidate PCFGs. Therefore, it is only useful for choosing between PCFGs. In this work, we propose a simple statistical method to solve detection problems: to determine whether or not a data set matches with a given PCFG, or to determine if two data set are generated from the same source. This approach is explained in detail in Chapter 3.

1.3 Applications

With these tools, we use traffic timing data to analyze three applications. The first application is to learn data structures for detecting protocols tunneled through Tor (the onion router) network. Tor is a commonly used anonymity tool on the Internet. From inter-packet timings of two systems which are communicating through Tor based on given protocols, we attempt to infer HMMs. However, the inferred HMM has a lot of noise events and the underlying system is difficult to discern. Using a Markov metric that we developed, we can identify the underlying model. Also, the metric measures the noise in the Tor communication. This application is interesting, since it considers models extracted from a system that is intentionally trying to obfuscate its internal workings.

The second application is centralized botnet traffic detection using HMMs. A centralized botnet is a collection of computers compromised by malware, and controlled by a centralized command and control (C&C) server for malicious purposes [44]. We focus on Zeus botnets (Zbots), one of the largest centralized botnets. Inter-packet timings of the botnet traffic relate to botnet activity characteristics (e.g. command and control processing time, idling time and contacting period), and

[3]PCFGs extend CFGs in the same way as HMMs extend regular grammars.

communication patterns among bots are similar. Therefore we infer HMMs from botnet traffic timing data. The inferred HMM is then used to detect whether or not a new observed traffic is from a botnet. Experiment results show that we can accurately detect botnet traffic.

Since centralized botnets (e.g. Zbot), can be detected using HMMs [54, 55]. And in centralized botnets, a bot communicates with the C&C sever only. Therefore, once one bot is detected, the centralized C&C server can be found and intercepted. And if the sever is taken down, the whole botnet is disabled. To make botnets more resilient, hierarchical botnets use peer-to-peer (P2P) techniques [5, 56, 66]. In P2P botnets, there is no centralized server and bots communicate with each other. Therefore their communications have recursive patterns. HMMs failed to detect hierarchical botnet traffic, because they can not represent recursive patterns [56, 64]. To solve this problem, we use PCFGs for P2P botnet traffic detection. With our PCFG approach, we detect the traffic from Storm botnet, one of the early P2P botnets [43]. The result shows that we can adequately distinguish the Storm botnet traffic from the normal traffic.

Our detection approach works with timing profiles, which exist in all automated network processes. Remove timing profiles is difficult and would have significant performance implications. Therefore it can be easily extended to detect traffic data from other network processes as well.

1.4 Organization

The remainder of this thesis is organized as follows:

- In Chapter 2, we introduce relevant background of this work, including probabilities, statistics, definitions of hidden Markov models and probabilistic context-free grammars. We also provide some background on Tor (the onion router) network and botnets.

- In Chapter 3, we introduce our pattern recognition approaches using stochastic grammars. We explain how to infer HMMs from data sequences and how to detect new observation sequence using confidence intervals (CIs) of HMMs. Markov metric is defined in this chapter. Some examples are given to show its performance. We also compare our metric with another commonly used distance measure for HMMs. Detection approach for PCFGs is then described, with illustrative examples and comparison with existing algorithms.

- In Chapter 4, we describe three applications using these tools. The first is the analysis on

Tor network data with Markov metric. We then use HMMs to detect real-world Zeus botnet traffic. With PCFGs, we can differentiate P2P botnet traffic from normal traffic. We give details about the experiment set-up and analysis of the results.

- We conclude this thesis with a summary of the achievements in Chapter 5. We also suggest extensions for future work.

Chapter 2

Background

This chapter provides background on our detection approaches. From observations, we infer production rule probabilities. Our approaches use standard statistical tests. So we provide an introduction to probability and statistics. We then define the stochastic grammars we use: hidden Markov models and probabilistic context-free grammars. Background on the Tor anonymity tool and botnets is also provided.

2.1 Probability and Statistics

This section provides some basic concepts from probability and statistics. Not all topics will be covered and the reader is encouraged to refer to probability and statistics textbooks, such as [72], [91] , [11] and [50], for a more detailed and thorough discussion on these concepts.

2.1.1 Probability

The formal definition of probability is as follows [72]:

Definition 1. *There is a universal set of possible outcomes Ω and $A \in \Omega$ is called a event. $P(A)$ is the probability of the event A, with following axioms:*

- $0 \leq P(A) \leq 1$*;*
- $P(\Omega) = 1$*;*
- *If $A_1, A_2, ...$ are disjoint events ($A_i \cap A_j = \emptyset$ for $i \neq j$), then $P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A)$.*

One might consider the probability of event A is the chance that A happens among all possible outcomes Ω. In practice, we usually estimate the probability of A using Equation 2.1 [72].

$$\hat{P}(A) = \frac{\text{Number of times A occurs}}{\text{Total number of all outcomes}} \tag{2.1}$$

If we use n to represent the total number of outcomes, then $P(A) = \lim_{n\to\infty} \hat{P}(A)$.

A *random variable* X is a real number associated with a random experiment [72]. A *discrete random variable* can take only a finite number (or a countably infinite number) of possible values [72]. It is described by a *probability distribution function* $p(x)$ [72], where:

1. $P(X = x) = p(x) \geq 0$
2. $\sum_x P(X = x) = 1$

A *continuous random variable* can take any value on some interval $[a, b]$, [72]. A *probability density function* $f(x)$ is used to describe a continuous random variable [72]:

1. $f(x) \geq 0$
2. $P[a \leq X \leq b] = \int_a^b f(x)dx$
3. $\int_{-\infty}^{+\infty} f(x)dx = 1$

The *Expectation* and the *Variance* are two commonly used features of random variables. The expectation $E[X]$ is the expected value of a random variable X:

$$E[X] = \sum_x xp(x) \tag{2.2}$$

$$E[X] = \int_{-\infty}^{+\infty} xf(x)dx \tag{2.3}$$

where Equation 2.2 is for discrete random variables and Equation 2.3 is for continuous random variables [72]. The expectation is also called as *mean* μ.

The variance $Var(X)$ measures the variability of the random variable X [72]:

$$Var(X) = \sigma^2 = E[(X - E[X])^2] = E[X^2] - E[X]^2 \tag{2.4}$$

The standard deviation is known as $\sigma = \sqrt{Var(X)}$ [72].

We introduce two continuous probability density distributions, the *normal distribution* and χ^2 *distribution*, which are used later in our detection approaches. The normal distribution, denoted as $N(\mu, \sigma^2)$, has the following probability density function [91]:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{2.5}$$

Its expectation is μ and variance is σ^2. It has a symmetric bell-shaped curve, as shown in Figure 2.1. The standard normal distribution is a normal distribution with $\mu = 0$ and $\sigma^2 = 1$ [91].

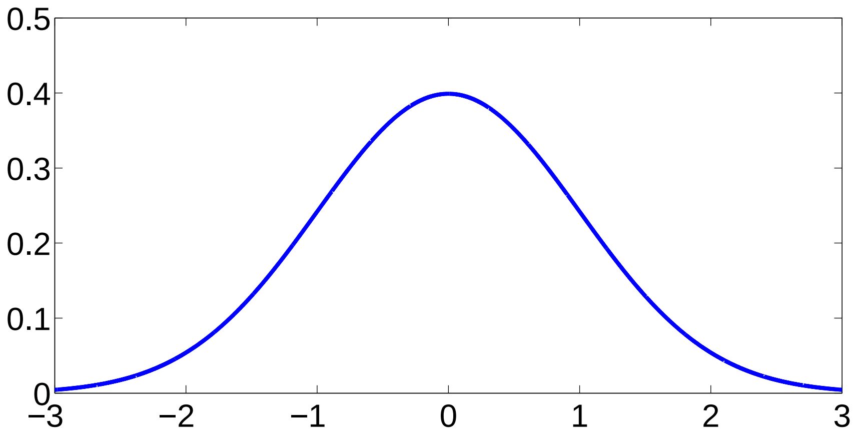


Figure 2.1: A normal distribution $N(0, 1)$

The cumulative distribution function for the normal distribution is [91]:

$$P[X \leq b] = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{b} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx \tag{2.6}$$

It is the area under the curve for $X \leq b$.

If $Z_1, Z_2..., Z_k$ are independent, standard normal random variables, the sum of their squares $X = \sum Z_i^2$ follows a χ^2 distribution with k degrees of freedom (DOF) [91]. It is a right-skewed curve with $X \geq 0$, as shown in Figure 2.2.

A standard statistical table or calculation software (e.g. R, MATLAB) can be used to find the cumulative distribution function for the χ^2 distribution.

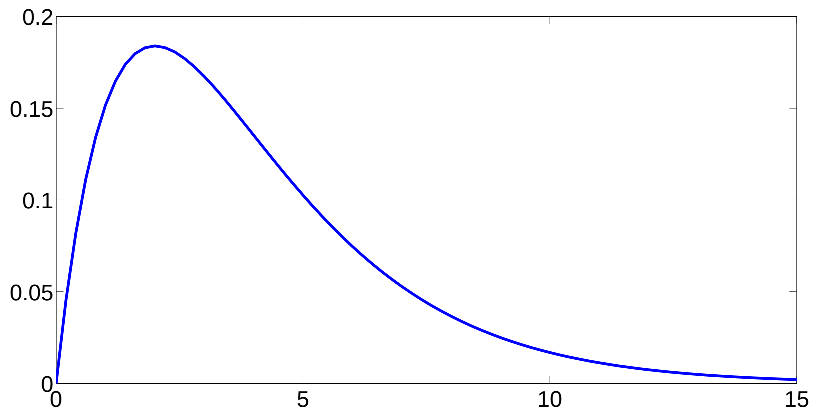


Figure 2.2: A χ^2 distribution with DOF=4

2.1.2 Statistics

In practice, since the whole population of the random variable is usually not available, statistics are calculated from a subset selected from the whole population (sample) to estimate the population parameters (mean, variance, etc.) [11]. Sample mean can be used to estimate the population mean, using [11]:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{2.7}$$

And to estimate the population variance, the sample variance is calculated using [11]:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 \tag{2.8}$$

These estimations are point estimates, which are usually not exactly equal to the population parameters. Therefore, in some cases interval estimates with a level of confidence are preferable [91]. A *confidence interval (CI)* is an interval estimate of a population parameter and indicates the confidence of the estimate [91, 11].

For a population parameter θ, if $P(\hat{\theta}_L < \theta < \hat{\theta}_R) = 1 - \alpha$, interval $(\hat{\theta}_L, \hat{\theta}_R)$ is then called a $(1-\alpha)100\%$ confidence interval for θ [91]. It means that we are $(1-\alpha)100\%$ confident that the true value of θ is in that interval [91]. The general form of a confidence interval (for symmetric distributions) is:

$$\text{Point estimate } \hat{\theta} \pm \text{Critical value} \times \text{Estimated standard deviation} \tag{2.9}$$

where the critical value is the percentile of the standardized value with $(1-\alpha)100\%$ confidence [11]. Figure 2.3 shows the percentiles for the standard normal distribution.

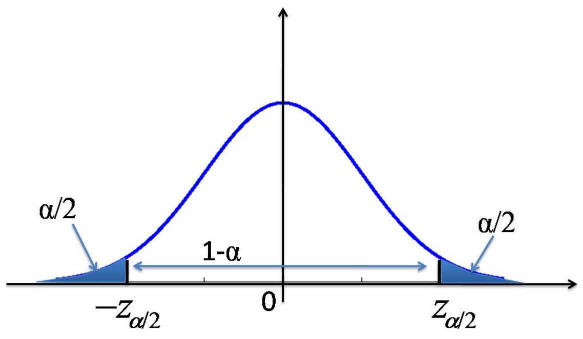


Figure 2.3: Percentiles ($\pm z_{\alpha/2}$) for the standard normal distribution

Statistical hypothesis tests are usually used to make decisions based on statistics calculated from data samples [11]. The *null hypothesis* H_0 is the hypothesis of interest. A predefined significance level α is required for the test. From the sample data, we calculate the appropriate test statistic. Comparing this statistic with the critical value from standardized distribution with the significance level α, we are able to make decisions to accept or reject the null hypothesis H_0 [11]. Figure 2.4 shows an example of the χ^2 test with the accept region and the critical value $\chi^2_{\alpha,4}$ in a χ^2 distribution with 4 DOF.

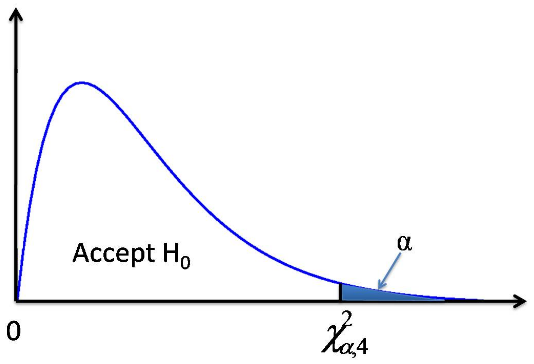


Figure 2.4: The critical value for a χ^2 test

A *Type I error* is an error that we reject null hypothesis H_0 when H_0 is true [91]. The probability of a Type I error is the significance level of the test α, which means the test will produce an $\alpha \times 100\%$ Type I error rate even when H_0 is true [91]. We can see this in Figure 2.4 as well.

Even if the calculated sample statistic follows a χ^2 distribution, it still goes beyond the critical value $\chi^2_{\alpha,4}$ for $\alpha \times 100\%$ of the time (filled area under the curve). Therefore α should be set to a Type I error rate that the test designer can tolerate [91]. A *Type II error* is an error that we accept the null hypothesis H_0 when H_0 is false [91]. The probability of a Type II error is β. Table 2.1 shows possible situations in statistical hypothesis tests [91].

Table 2.1: Statistical Hypothesis Testing Results

	H_0 is true	H_0 is false
Accept H_0	Correct decision $(1-\alpha)$	Type II error (β)
Reject H_0	Type I error (α)	Correct decision $(1-\beta)$

2.1.3 ROC Curves

We develop our detection approaches for stochastic grammars based on statistical tests. With some testing data, we can check the performance of our approaches. Table 2.2 shows all possible results from detections, and it is also called a confusion matrix [23].

Table 2.2: Confusion Matrix

Actual / Detection	Positive	Negative
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

To evaluate the performance of a detection approach, several measures are derived from the confusion matrix. The true positive rate (TPR), calculated by $TPR = TP/(TP + FN)$, evaluates the ability of the detection approach to find the correct positive results among all positive data [23]. The false positive rate (FPR), calculated by $FPR = FP/(FP + TN)$, shows the ability of the detection approach to distinguish the positive data from the negative data [23]. A good detection approach generally means high TPR and low FPR. However, pursuing one goal usually sacrifices the other. A receiver operating characteristic (ROC) curve is then used to evaluate the overall performance of a detection approach while tuning some parameters in the detection [23]. It plots the true positive rates vs. the false positive rates. An example of ROC curve is shown in Figure 2.5.

The optimal point (0,1) in the ROC plot represents 0% false positive rate and 100% true positive rate, which is the ideal detection rate for all detection approaches [23]. In practice, the detection ROC curve usually can not achieve this optimal point, and is similar to the curve in

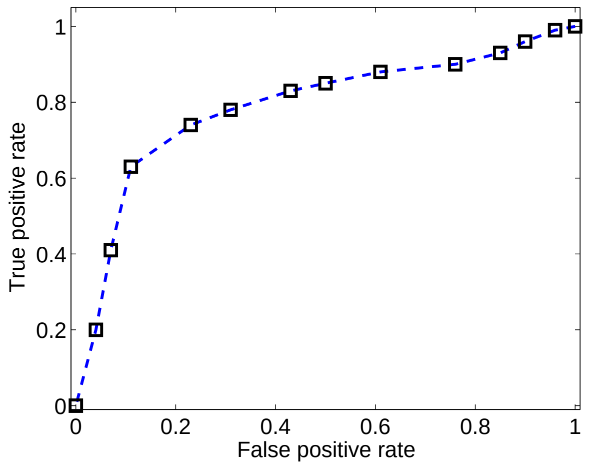


Figure 2.5: An example ROC curve

Figure 2.5. To considers the trade-off between TPR and FPR, we use the closest (in the Euclidean metric) point to the optimal point (0,1) in the curve [22]. The corresponding TPR and FPR are the best detection rates of the approach [22].

2.2 Hidden Markov Models

A Markov model is a tuple $G = (V, E, P)$ where V is a set of vertices of a model, E is a set of directed edges between the vertices, and P is a probability matrix such that:

$$\sum_{v_j \in V} p(v_i, v_j) = 1, \quad \forall v_i \in V \tag{2.10}$$

A Markov model satisfies the Markov property, which means that the next state of the process depends only on the current state. It is also known as "memoryless".

A hidden Markov model (HMM) is a Markov model with output labels and assumes the states are hidden [70]. A standard HMM has two sets of random processes, one governing state transition and the other governing symbol outputs. In this paper, we use another representation which has only a single random process: the state transitions. The output symbols are associated with transitions. This representation is equivalent to the standard model [15]. It is possible to create models satisfying our constraints from the classical representations [70, 15]. We use this representation because it is simple yet does not lose any expressive power.

This representation uses a tuple $G = \langle \mathcal{A}, V, E, P \rangle$, where $\mathcal{A}$ is a finite alphabet, V is a finite set of nodes or states, $E \subseteq V \times \mathcal{A} \times V$ is a transition relation, and $P : E \rightarrow [0, 1]$ is a probability function such that

$$\sum_{a \in \mathcal{A}, v_j \in V} p(v_i, a, v_j) = 1, \quad \forall v_i \in V \tag{2.11}$$

The vertices of G are referred to as states and edges as transitions, where V is the state space of size m. The $m \times m$ transition probability matrix P can be constructed for G. Each element $p_{i,j} \in P$ expresses the probability the process transitions to state v_j once it is in state v_i. A *path* through G with label $\mathbf{a} = a_1 a_2 \cdots a_n$ is an ordered set of vertices $(v_1, v_2, \ldots, v_{n+1})$ such that for each pair (v_i, v_j):

1. $p(v_i, v_j) > 0$; and
2. $E(v_i, v_j) = a_i$.

We require E to be a deterministic relation such that for $E(v_i, v_j) = a$ then $E(v_i, v_k) \neq a$ for all $v_i, v_j, v_k \in V$. The probability that a certain symbol a will occur next is the probability p_a of moving to the next state using a transition associated with the symbol a. If no transition exists between state v_i and state v_j, $p(v_i, v_j) = 0$ and $E(v_i, v_j)$ is undefined. Figure 2.6 shows an example HMM.

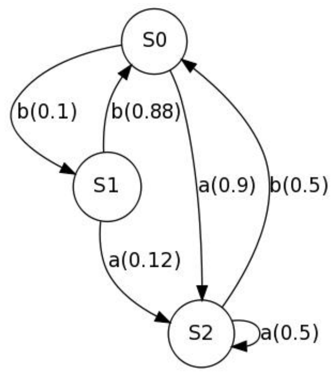


Figure 2.6: An example HMM

We assume that transition probabilities are constant, i.e. each element $p(v_i, a, v_j)$ does not change over time. The asymptotic state probabilities (steady state probabilities) is the stationary distribution of the HMM. For *ergodic* Markov processes, which do not have absorbing states (i.e. has no outgoing transitions), the asymptotic state probabilities $\vec{S} = (s_1, s_2, .., s_m)'$ can be calculated from:

$$\begin{cases} P\vec{S} = \vec{S}, \\ \sum_i^m s_i = 1 \end{cases} \tag{2.12}$$

The first equation is the definition of the stationary property. The second equation is the constraint of the sum of all probabilities. Solving the equations for $\vec{S}$, we can get the asymptotic state probabilities of the HMM.

A series of successive symbols representing a path of transitions is called an observed output sequence. An output sequence of length l can be generated from a HMM with the following procedure:

i. Randomly choose an initial state $v_i = v_0$ based on asymptotic state probabilities,

ii. Using the probabilities of the outgoing transitions, select a transition $p_{i,j}$ to move to state v_j from state v_i

iii. Record the label $a_i = E(v_i, v_j)$ associated with chosen transition $p_{i,j}$

iv. Repeat steps *ii* and *iii* until l labels have been recorded

v. Record the sequence $\mathscr{A} = a_i a_{i+1} a_{i+2} \cdots a_{i+l-1}$

We note that if the model contains at least one absorbing state, it is possible for the model to be unable to generate a sequence of length l. In this paper we restrict our discussion to *ergodic* Markov processes, which always have non-trivial asymptotic state probabilities.

2.3 Probabilistic Context-Free Grammars

Although HMMs are widely used for pattern recognition and detection, they are the simplest class (regular grammar) in the Chomsky Hierarchy of formal grammars [64]. In this section, we first discuss grammars and the Chomsky Hierarchy.

A grammar is a tuple $G = (N, \Sigma, R, S)$ where [13]:

1. N is a set of nonterminal symbols,

2. Σ is a set of terminal symbols,

3. R is a set of production rules in the form of $\alpha \rightarrow \beta$, where $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$ and $\beta \in (N \cup \Sigma)^*$ [1],

4. S is the start symbol, which is in N.

In this thesis, we use standard notation for grammars. As a convention, we use $A, B, \ldots$ to denote nonterminal symbols, $a, b, \ldots$ to denote terminal symbols and $\alpha, \beta, \ldots$ to denote strings in $(N \cup \Sigma)^*$.

The Chomsky Hierarchy, shown in Figure 2.7, defines four levels of grammars based on the production rules used [64]. The simplest is the regular grammar, where production rules have the format: $A \rightarrow a$ or $A \rightarrow aB$, where $A, B \in N$ and $a \in \Sigma$ [64]. For a HMM, a state is equivalent to a nonterminal, transition labels are terminals and transitions between states are production rules, which always have format $A \rightarrow aB$. A HMM is a probabilistic regular grammar because its production rules (transitions) are associated with probabilities.

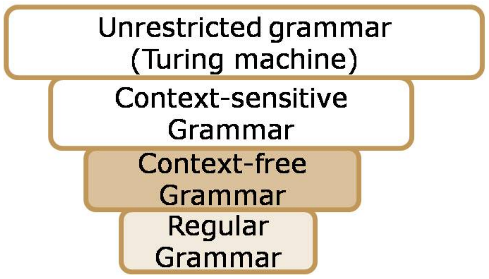
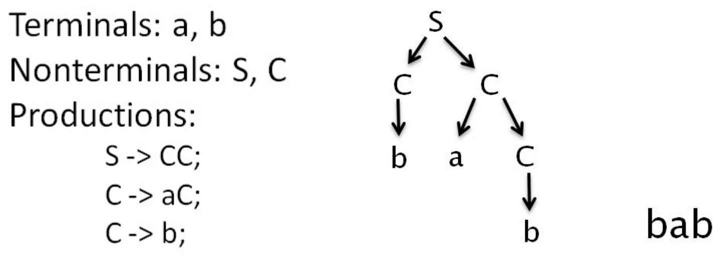

Unrestricted grammar (Turing machine)
Context-sensitive Grammar
Context-free Grammar
Regular Grammar

Figure 2.7: Chomsky Hierarchy

HMM can only represent patterns in regular expressions and can not detect recursive patterns, such as $a^n b^n$ (This means a sequence with equal number (n) of a's and b's, i.e. $aabb$, $aaabbb$) [13]. We need context-free grammars (CFGs) to recognize or detect recursive patterns. A CFG is a grammar with production rules in the form of $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$ [64]. The right-hand side of a CFG production can be any combination of terminals and nonterminals. Production rules of regular grammars also satisfy CFGs constraints, but productions of CFGs may not satisfy regular grammar constraints. Therefore the set of regular grammars is a proper subset of CFGs. CFGs represent a larger class of patterns than HMMs, including recursive patterns like $a^n b^n$.

[1] $(N \cup \Sigma)^*$ means 0 or more combinations of terminals and/or nonterminals

From the start symbol, a CFG can generate a *tree* using production rules until all the leaves in the tree are terminals. Based on depth-first search, a tree corresponds to a *sentence*, which is a sequence of terminals. $T(G)$ is used to represent the set of trees that can be generated by G, and $L(G)$ is used to represent the set of sentences that can be generated by G. Figure 2.8 shows a simple example of a CFG, a tree generated by the CFG and the corresponding sentence of the tree.



(a) CFG Example (b) Example Tree (c) Sentence

Figure 2.8: Example of CFG tree and sentence

A probabilistic context-free grammar (PCFG) is a CFG where each production rule is augmented by a probability, which can be denoted as $p(A \rightarrow \alpha)$. PCFGs extend CFGs in the same way as HMMs extend regular grammars. A PCFG is *proper* if $\sum_\alpha p(A \rightarrow \alpha) = 1$ is true for every nonterminal A [27]. A PCFG is called *consistent* if no probability mass is lost in generating infinite size trees or sentences, i.e. $p(T(G)) = 1$ [27]. The *branching rate* is used to describe the geometric growth rate of the tree or sentence derivations [24, 60]. A branching rate that is larger than 1 may result in a tree or sentence with infinite size, making the PCFG inconsistent [24], which means that this PCFG has non-zero probability of generating trees with infinite sizes. Since we limit our approach to finite trees and finite sentences, we only consider PCFGs with branching rate less than 1, that are consistent.

PCFGs map language memberships to probability distributions. Noise and pattern distortion can be modeled using low probabilities. Therefore, we concentrate more on detecting PCFG production probabilities than determining language membership [51]. We assume production rules are known and data sets are languages of the grammar in this paper.

2.4 The Onion Router (Tor)

Tor [6] is one of the most popular anonymization systems in use today. Tor computers run three types of services: relay, directory server, or client. Each computer can run one or more services. Relays transfer data from clients to other relays, between two relays, or retrieve external resources for a client. A list of available relays, commonly known as consensus, is published by pre-defined directory servers on a TCP port where it can be downloaded by clients. Each data packet sent through Tor is iteratively encrypted using the key of each successive relay of the circuit. The result is similar in nature to an onion. As each relay in the circuit receives the onion packet, it will decrypt and peel away a layer, forwarding on what remains. Figure 2.9 shows the packet transmission process in Tor network. In this example, Alice sends packets to Bob through 3 Tor nodes. No intermediate node knows both Alice and Bob, and anyone eavesdropping on the communications does not know both ends either. In this way, Tor provides anonymity for both Alice and Bob. For more details of the process, see [30].

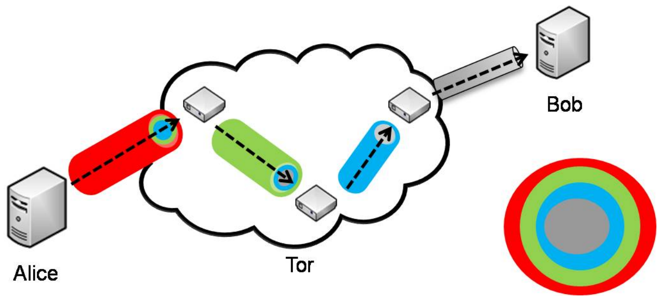


Figure 2.9: Tor communications

Tor is designed to obfuscate communications patterns, however, since Tor is a low-latency system that does not disrupt the timing of the packets as they propagate the network, the traffic patterns, especially timings, are similar among different relay nodes, except for some jitter (noise). Even though different relays use different encryption keys, their timing information is not changed by encryption. Therefore we can attempt to identify the communication protocol tunneled through Tor from observed inter-packet delays.

2.5 Botnet

A botnet is a collection of computers compromised by malware, referred to as bots, and controlled by a command and control (C&C) server for sending spam messages, performing distributed denial of service attacks, stealing login credentials, or stealing personal information. Botnets are a major threat to Internet security [96] [93].

2.5.1 Centralized Botnet

Botnet communication usually uses a star topology, where a single centralized C&C server remotely controls all bots, as shown in Figure 2.10 [66]. After a computer is infected, it will automatically announce itself to the C&C server and get instructions from it. The C&C channel is the critical part of a centralized botnet, since it is the way a botherder (the hacker who controls the botnet) directs the botnet. Furthermore, C&C communications among bots follow a well-defined protocol. Detection of C&C communications is important for identifying and stopping the botnet.

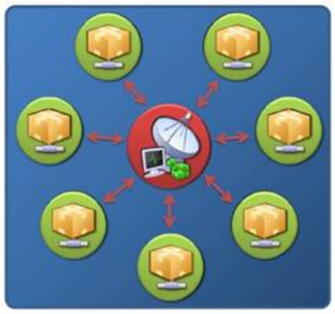


Figure 2.10: Centralized botnet [66]

Many botnets in the wild use IRC or HTTP for C&C communication [44]. When control is managed via IRC, the C&C server pushes commands to bots, so the C&C has instant control of the bots. While for HTTP, the C&C update commands wait for bots to fetch them. Bots repeatedly contact the C&C server to get commands, so the C&C has delayed control of the botnet. In both cases, command and control of the botnet is centralized. Communications between bots and C&C servers are usually encrypted, making it more resilient to analysis and detection.

Botnet traffic data has identifiable patterns in timings. Bots repeatedly communicate with the C&C server to update their configurations or get new commands. Inter-packet timings relate to command execution time, idling time, contact period and other botnet activities, and therefore are a consequence of botnet behaviors. Inter-packet timing data can be obtained easily by network sniffing without reverse engineering malware binaries or encrypted packets, and different bots of the same botnet will have similar communication patterns. For example, bots send similar activity messages or execute the same command at a similar time intervals. Therefore, we deduce patterns in inter-packet delays with stochastic grammars, and use them to detect similar botnet traffic.

2.5.2 P2P Botnet

For centralized botnets, if a bot agent is found and intercepted, the central C&C server can be identified. Once the centralized C&C server is taken down, the whole botnet is disabled. To avoid this, P2P technologies have been used to create hierarchical botnets [5], as shown in Figure 2.11 [66]. In a hierarchical botnet, bot agents send C&C instructions from their parent node to children nodes that they infected. So most bots are not aware of the C&C server of the botnet and detected bots are unlikely to reveal the C&C server [66].

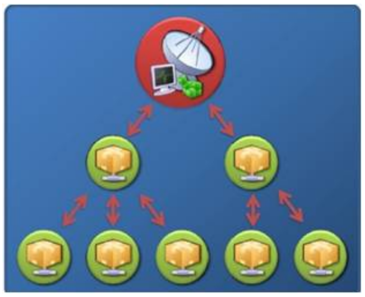


Figure 2.11: P2P hierarchical botnet [66]

In a P2P botnet, since every node includes timing patterns of its neighbor (parent/children) bots, different bots may have quite different communication patterns, because some bots may have few neighbor bots while others may have a lot of neighbor bots. The communications also have recursive patterns because of the hierarchical structure of the P2P botnet. However, their communications may have similar or equivalent probability distributions for malicious actions. For example, if the botnet is performing DDoS attacks, a large portion of the bot's traffic will be fast and repeated packets, and these characteristics among bots are usually similar no matter how many neighbors a bot may have. Based on these observations, we develop detection approaches with more expressive power grammars (PCFGs) to detect P2P hierarchical botnet traffic patterns, because they can represent hierarchical and recursive patterns.

2.6 Summary

In this chapter, we gave essential background information for this thesis, including probability definitions, statistical distributions (the normal distribution and χ^2 distribution), basics about statistical tests that are used for our detection approaches, and detection performance analysis using ROC curves. We also introduced two stochastic grammars: hidden Markov models and probabilistic context-free grammars. Finally, a brief discussion of the Tor network and botnets, which are used in our applications, is provided. We also discuss motivations for these applications of stochastic grammars, based on the underlying characteristics of these network processes.

Chapter 3

Pattern Detection

In this chapter, we describe our pattern detection approaches using hidden Markov models (HMMs) and probabilistic context-free grammars (PCFGs). Using previous work, we infer HMMs from a sequence of data observations without *a priori* knowledge of the HMM structure and parameters. We then extend the confidence interval (CI) approach in [22] to detect whether or not a new sequence matches the HMM. Using ROC (Receiver Operating Characteristic) curves, we can find the optimal detection rate. Similar data sets can create models that represent the same system with minor differences in probability or structure. It may, however, be difficult to tell if model differences are minor or significant. To compare HMMs, we propose a normalized Markov metric. Instead of concentrating on the structure of the HMMs, our metric is based on the system statistics that the HMM represents. For PCFGs, we propose a simple statistical approach for pattern detection. After estimating production probabilities from data sets, we design a χ^2 test as a criteria for pattern detection with a level of statistical significance. This chapter describes these approaches in detail and some illustrative examples are used to show the performance of them.

3.1 Detection with HMMs

This section introduces our HMM inference and detection method. We infer HMMs from data without *a priori* information [76]. A model confidence test is used to ensure enough data is used [97]. With inferred HMMs, we extend the CI approach in [22] to detect new observation sequences. The detection results are shown in ROC curves, which consider trade-off between true positive and false positive rates.

3.1.1 HMM Inference

HMM inference discovers the HMM structure and state transition probabilities from a sequence of output observations. Traditionally, the Baum-Welch algorithm is used to infer the state transition matrix of a Markov chain and symbol output probabilities associated to the states of the chain, given an initial model and a sequence of symbols (see [70]). As a result, this algorithm requires the *a priori* structural knowledge of the Markov process that produced the outputs.

In [78, 77, 80, 79], an approach is developed that derives the HMM state structure and transition matrix from available data without knowing the *a priori* structure of the HMM. Given a sequence of symbolic data, it produces several sets of causal states which contain subsequences of the data, and a transition structure of a HMM. This approach needs a parameter L, which is large enough that the history sequence with length L is sufficient to indicate the state where the process is in. With this assumption, the history sequence with length less than or equal to L are grouped into different states, by comparing the conditional probability of the next symbol. This detailed approach is provided in Appendix C.

In our work, we use an extension [76] to determine the parameter L and therefore derive HMMs with no *a priori* information. We increase L, and infer a HMM for each L. A symbol-to-state mapping is built to check if the HMM structure stabilizes. If the inferred HMM stabilizes, we declare the model correct [76]. This occurs because with larger L, no additional statistical relevant information will be gained by the original approach. This approach is described in Appendix C.

If an insufficient amount of observation data is used to generate the HMM, the model will not be representative of the actual underlying process. We use a model confidence test to determine if the observation data and constructed model fully express the underlying process with a given level of statistical significance [97]. This approach calculates an lower bound on the number of samples required. If the number of input samples is less than the bound, more data is required. New models should be inferred with more data and still need to be checked for the confidence. This approach allows us to remove the effect of noise data in the HMM inference. The HMM inference process is shown in Figure 3.1.

We use a simple example to illustrate the process of HMM inference. This example data set contains 2,000 symbols (symbolized traffic timing data captured in a local Zeus botnet). We

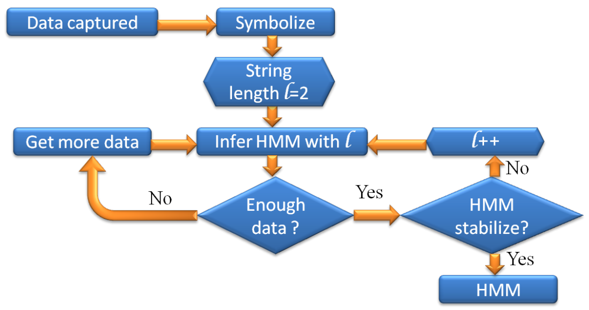
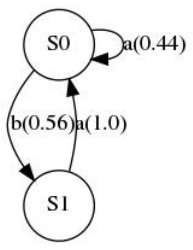
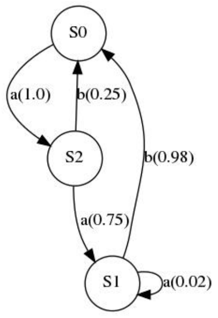


Figure 3.1: HMM Inference Process

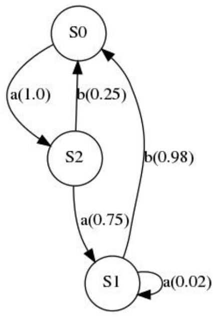
inferred HMMs from it and checked the model confidence. In Figure 3.2, we show inferred models with different L and results from the model confidence test. After $L = 3$, the inferred HMM stabilizes, further, the used data size is larger than the required size. Thus we get the statistical significant HMM. This is shown in Figure 3.2.



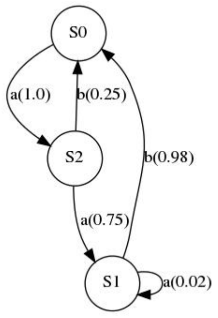
(a) $L = 2$, required samples:286



(b) $L = 3$, required samples:372



(c) $L = 4$, required samples:372



(d) $L = 5$, required samples:372

Figure 3.2: Inferred HMMs from example Data

3.1.2 HMM Detection

Once the HMM is inferred from symbolized data and passes the model confidence test, it can be used to detect botnet traffic. The traditional Viterbi Algorithm [70] finds the HMM that was most likely to generate the data sequence by comparing probabilities generated by the HMMs. For data streams, it is unclear what sample size to use with the Viterbi algorithm. Also as data volume increases, the probability produced by the Viterbi algorithm decreases exponentially, and may suffer floating point underflow [22]. To remedy this, the confidence intervals (CIs) approach was developed [22]. With this approach, the certainty of detection increases with the number of samples and the floating point underflow issues of the Viterbi Algorithm eliminated [22]. We use the CI approach to determine whether or not the observed network traffic matches the botnet HMM.

Given a sequence of symbolized traffic data and a HMM, the CI method in [22] traces the data through the HMM and estimates the transition probabilities and confidence intervals. This process maps the observation data into the HMM structure. It then determines the proportion of original transition probabilities that fall into their respective estimated CIs. If this percentage is greater than a threshold value, it accepts that the traffic data adequately matches the HMM. In this paper, we take an alternative approach. Instead of estimating transition probabilities and corresponding CIs, we calculate the state probabilities, which are the proportion of time the system stays in a specific state, and their corresponding CIs. This estimation approach is described in Algorithm 3.1.1.

```
Algorithm Description 3.1.1 – Estimate State Probabilities and CIs
Input: HMM $G$; A sequence of observations $\mathscr{A}$ with length $N$.
  1. For every state $v_k$ in $G$, use $v_k$ as start state to estimate state probabilities:
      (a) Initialize counters $c_i$ for each state $v_i$ in model $G$ to zero.
      (b) Starting at $v_k$ and follow the path defined by transitions $(v_i, v_j)$ associated with each symbol
          $a_i \in \mathscr{A}$ in turn. Since the HMMs we used are deterministic, this path is unique.
            i. If there is no transition associated with the corresponding symbol in $\mathscr{A}$, stop considering
               the path. It could not have been generated by $G$ starting at state $v_k$;
           ii. Otherwise add one to counter $c_i$ for state $v_i$;
          iii. Estimate the probability of state $v_i$ as $\hat{s}_i = \frac{c_i}{N}$;
  2. If it was found that no start state could have generated sequence $\mathscr{A}$, then exit with no matching.
  3. For state $v_k$ where all symbols in $\mathscr{A}$ are accepted by $G$ using $v_k$ as a start state, calculate CIs for
     estimated state probabilities using:
         $[s_i - Z_{\alpha/2}\sqrt{s_i(1-s_i)/N}, s_i + Z_{\alpha/2}\sqrt{s_i(1-s_i)/N}]$        (3.1)
     where $s_i$ is the asymptotic (steady) state probability of state $v_i$ and $Z_{\alpha/2}$ is from the standard normal
     or t distribution.
```

Once all state probabilities and CIs are estimated for the observation sequence, we determine for each state v_i whether or not $\hat{s}_i$ is within the confidence interval in Equation 3. If it matches, we accept the null hypothesis that $\hat{s}_i = s_i$ with a probability of false positives equal to α. For the whole HMM, we can then get the proportion of states whose estimated state probability $\hat{s}_i$ matches the corresponding confidence intervals. If an observation sequence is generated from the HMM, it will follow the state transitions of the underlying stochastic process that the HMM represents, and its state probabilities will converge to the asymptotic state probabilities if the sequence length is large enough. Therefore, more states will match their estimated CIs. Generally, a sequence that is generated by the HMM will have a high proportion of matching probabilities, while a sequence that is not an occurrence of the HMM will have a low proportion of matching probabilities. Similar to the detection approach in [22], a threshold value can be set for this proportion of matching states, to determine whether the observation sequence matches with the HMM.

In [22], receiver operating characteristic (ROC) curves are used to find the optimal threshold value. By varying the threshold from 0% to 100% (0% threshold means we accept everything and we will have a high false positive rate. 100% threshold means we reject everything and we will have a low true positive rate), we progressively increase the criteria for acceptance. Using the ROC curve drawn from detection statistics with different thresholds, we find the closest point to (0,1), which represents 0% false positive rate and 100% true positive rate. This considers the trade-off between true positive and false positive rates. Therefore, the corresponding threshold of that point is the optimal threshold value.

Figure 3.3 gives an example of detection. After tracing a sequence of data through the original HMM in Figure 3.3-(a), we get the resulting HMM in Figure 3.3-(b) with estimated transition, state probabilities, and confidence intervals. Since all steady state probabilities of the original HMM fall into corresponding estimated CIs, the percentage of matching is 100%. By checking the detection ROC curve in Figure 3.3-(c), we find the optimal point is $(0.07, 0.95)$, and corresponding threshold value for the percentage of matching state probabilities is 67%. Since $100\% > 67\%$, we declare this sequence data matches the original HMM.

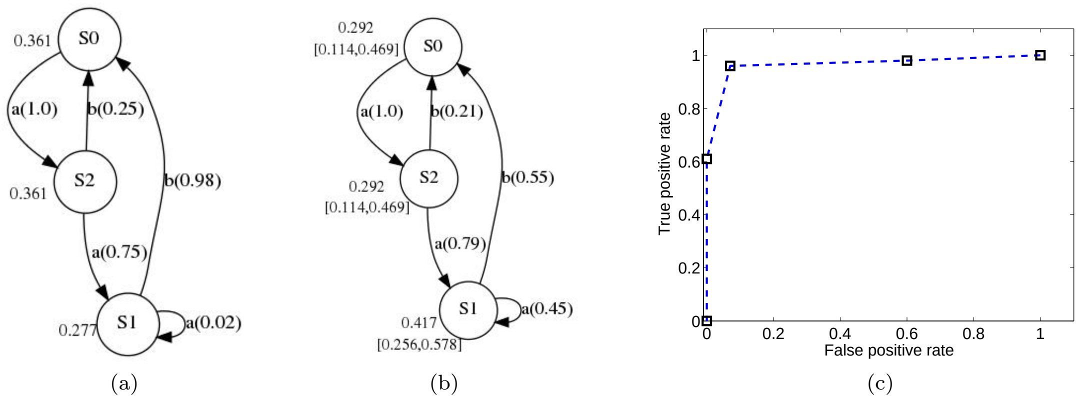


Figure 3.3: HMM Detection: (a) Original HMM (steady state probabilities are shown next to states), (b) CI HMM (confidence intervals are under steady state probabilities), (c) Detection ROC curve (95% CI)

3.2 Markov Metric

With HMM inference approaches, similar data sets can create models that represent the same system with minor differences in probability or structure. It may, however, be difficult to tell if model differences are minor or significant. Therefore, a metric is useful for determining if one model is simply a different representation of the same process, a minor variation thereof, or a representation of a very different system. A metric is a mathematical construct that describes the similarity (or difference) between two models. For example, it is useful to know if two processes are the same except for rare events (e.g., events occurring once in a century when measuring daily rainfall), since we would typically consider them functionally equivalent. Eliminating duplicate models can reduce system complexity by decreasing the number of models to analyze using observation data. Grouping similar models can increase the number of samples available for model inference, leading to higher fidelity system representations.

The problem we consider is determining if two models represent the same or related Markov processes, or alternatively, if the processes they represent are similar. We solve this problem by deriving a metric space for comparing Markov processes and their associated models. Our approach works on Markov processes as long as a few assumptions are true: the processes must have a finite state space, they must be ergodic so that non-trival asymptotic state probabilities exist, and the transition probabilities must be stationary. These assumptions hold for the majority of hidden Markov models used in practical applications.

Since a hidden Markov model is a probabilistic graph[70], [40], it is tempting to measure model similarity by finding correspondences between states [71], like graph matching. This approach would work if the structures are simple and similar. In practice, however, models inferred from real data usually contain a large amount of noise, which could result in models with very different structures. Furthermore, graph matching is often intractable and impractical for graphs with a large number of vertices and edges [42].

Instead of using graph matching to determine if two HMMs are the same, we use statistical sampling to measure the similarity of HMMs by leveraging the statistics of the stochastic processes they represent. The same stochastic process can be represented by HMMs with very different structures. For example, the models in Figure 3.4 represent the same stochastic process, even though the two models shown have different structures. No matter which state the system is in, symbols A and B are equally likely to be generated. Graph matching algorithms will be unable to find correspondence between the graph structures and may erroneously find that the two models are not the same.

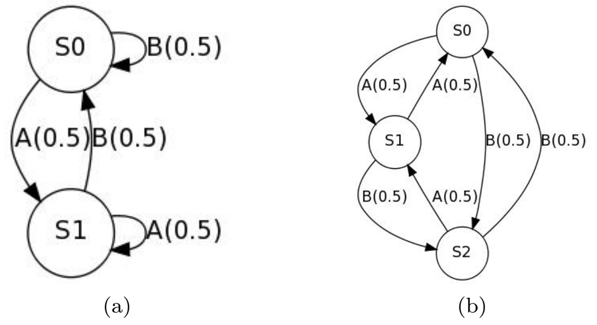


Figure 3.4: Two distinct HMMs represent the same stochastic process.

We propose a novel metric space for Markov processes based on the system statistics, which are determined by the stochastic process the HMM represents [57]. We use the χ^2 test in our comparisons to determine if models are equivalent within a given level of statistical significance. Our reliance on statistical testing requires us to introduce the parameter α, which explicitly states the degree of significance used for comparing distributions [63]. In addition, our metric uses data sampling insights to systematically remove states and transitions from HMMs to find a structure close to both models being compared, which expresses the elements of both models that are statistically equivalent. This structure defines the distance between two models.

We provide the comparison between our approach and Kullback-Leibler divergence [48], which is another widely used approach for measuring HMM similarity. Our approach is a true metric, can always return an appropriate distance value, and provides a confidence measure on the metric value.

3.2.1 HMM Equivalence

As shown in Figure 3.4, HMMs may represent the same underlying process with quite different state structures and transition probability matrices. We define equivalence ($G_1 \equiv G_2$), as G_1 and G_2 accepting the same symbol sequences with statistical significance α. To compare models for statistical equivalence, we extend the confidence interval approach in [22] to use the χ^2 test of equivalence for sets of normal distributions as shown in Algorithm 3.2.1.

Algorithm Description 3.2.1 – Hidden Markov Model Equivalence

Input: HMM G_1, HMM G_2, Sequence length N, Significance level α;

Main procedure:

1. Generate a sequence $\mathscr{A}$ of length N from G_1;
2. For every state v_k in G_2, use v_k as start state v_0 to calculate associated transition probability estimates $\hat{p}_{i,j}$ for each transition in G_2:
   - (a) Initialize counters c_i for each state v_i in model G_2 and counters $c_{i,j}$ for each transition leaving state v_i to zero.
   - (b) Starting at v_0 and follow the path defined by transitions (v_i, v_j) associated with each symbol $a_i \in \mathscr{A}$ in turn. Since the HMMs we used are deterministic, this path is unique.
     - i. If there is no transition associated with the corresponding symbol in $\mathscr{A}$, stop considering the path. It could not have been generated by G_2 starting at state v_0;
     - ii. Otherwise add one to counter c_i for state v_i and add one to counter $c_{i,j}$ for transition (v_i, v_j);
     - iii. Estimate the probability of transition (v_i, v_j) as $\hat{p}_{i,j} = \frac{c_{i,j}}{c_i}$;
3. If it was found that no start state could have generated sequence $\mathscr{A}$, then exit with no equivalence.
4. For every state v_k where all symbols in $\mathscr{A}$ are accepted by G_2 using v_k as a start state, construct ${G_{2,k}}'$ using the estimated transition probabilities from Step 2(b)iii.
5. Compute the χ^2 statistic χ^2_R for comparing model G_2 with models ${G_{2,k}}'$ using the method proposed in [85], shown in the next part;
6. Determine the χ^2_α statistic for the χ^2 test with significance level of α and $n_t - n_s$ degrees of freedom, where n_t is the number of transitions and n_s is the number of states in G_2. This can be done either by using standard statistical tables [63] or calculating the value directly [68];
7. If $\chi^2_R <= \chi^2_\alpha$ for any ${G_{2,k}}'$, the χ^2 test accepts with significance α the hypothesis that the probability density functions in G_2 are consistent with the probability density functions in that ${G_{2,k}}'$. Since ${G_{2,k}}'$ is calculated using a data trace from G_1, this means that their statistics are consistent. Exit with equivalence;
8. If $\chi^2_R > \chi^2_\alpha$ for all ${G_{2,k}}'$ then exit with no equivalence;

Statistic χ^2_R calculation from [85]:

1. Construct a vector $\vec{p} = \begin{pmatrix} \hat{p}_{1,1} - p_{1,1} \\ \ldots \\ \hat{p}_{i,j} - p_{i,j} \\ \ldots \end{pmatrix}$, where for each state i, we put $|v_{i,j}| - 1$ probabilities in the vector ($|v_{i,j}|$ is the number of transitions leaving state i). Therefore, there are a total of $n_t - n_s$ entries in vector $\vec{p}$;
2. Construct the covariance matrix Σ_p for vector $\vec{p}$. For each pair of entries $\begin{pmatrix} \hat{p}_{i,j} - p_{i,j} \\ \hat{p}_{i',j'} - p_{i',j'} \end{pmatrix}$ in $\vec{p}$, using the following equation to get the corresponding entries in Σ_p:
$$\begin{pmatrix} p_{i,j}(1-p_{i,j})/s_i & \delta_{i,i'}p_{i,j}(\delta_{j,j'}-p_{i,j'})/s_i \\ \delta_{i,i'}p_{i,j}(\delta_{j,j'}-p_{i,j'})/s_i & p_{i',j'}(1-p_{i',j'})/s_{i'} \end{pmatrix} \quad (3.2)$$
where $\delta_{i,j} = 1_{[i=j]}$ is the Kronecker delta indicator. In this matrix, the covariance entries $\delta_{i,i'}p_{i,j}(\delta_{j,j'} - p_{i,j'})/s_i = 0$ for $i \neq i'$;
3. The statistic is $\chi^2_R = N\vec{p}^T\Sigma_p^{-1}\vec{p}$.

In Algorithm 3.2.1 we use HMM G_1 to generate a string of data $\mathscr{A}$, summarizing the conditional probabilities of the model's Markov process [76, 78]. Sequence $\mathscr{A}$ is then run through HMM G_2, where we use frequency counting to construct a set of estimates ${G_{2,k}}'$ of the transition probabilities in G_2 [22]. The standard χ^2 test compares the distribution means to determine whether or not the two sets of transition distributions $({G_{2,k}}', G_2)$ are equivalent with significance α [63, 68]. Since ${G_{2,k}}'$ simply maps conditional probabilities from G_1 onto the structure of G_2, this shows by extension that $G_1 \equiv G_2$.

Remark 1. *The computational complexity of this algorithm is $O(Nn_s)$. The complexity of the χ^2 test ($O(n_t - n_s)$) is negligible, since the sequence length N is usually much larger than $(n_t - n_s)$. The multiplicator n_s in $O(Nn_s)$ denotes that all states will be tried as a start state before the algorithm finds the correct start state to accept the sequence.*

Remark 2. *The length of sequence $\mathscr{A}$ must be large enough to represent each transition probability asymptotically with a normal distribution. We discuss how to determine sequence length in Section 3.2.5.*

Remark 3. *The work in [85] shows that $\vec{p}$ follows a normal distribution $N(\vec{0}, \frac{\Sigma_p}{N})$. Therefore the χ^2 test we use is different from traditional χ^2 test, which usually calculates the statistic using $\chi^2_R = \sum_{i=1}^{n_s} \sum_{j=1}^{|v_{i,j}|-1} \frac{(c_i \hat{p}_{i,j} - c_i p_{i,j})^2}{c_i p_{i,j}}$. The traditional χ^2 test considers the covariances between every pair of probabilities in $\vec{p}$, while in our case, the covariances between probabilities of transitions leaving from different states is 0 ($\delta_{i,i'} p_{i,j}(\delta_{j,j'} - p_{i,j'})/s_i = 0$ when $i \neq i'$) [85]. Therefore we only need to consider the covariances between probabilities of transitions leaving from the same state. This is because of the Markovian property of HMMs.*

Remark 4. *For every state, probabilities of the outgoing transitions must sum to one, which removes one degree of freedom for each state from the χ^2 test. Therefore, the degrees of freedom for a model is $n_t - n_s$.*

Remark 5. *Usually we use the HMM with more degrees of freedom (DOF) as G_1 to generate the sequence of data and test it using the model with fewer degrees of freedom. Each probability in the smaller model will be inferred using a larger number of samples, so the observed probabilities should be closer to the true value. Detailed discussion is in Section 3.2.5.*

Remark 6. *Statistical significance α is required, as in all statistical tests, to represent the likelihood of Type I error that is acceptable. It means that we are willing to accept the null hypothesis (model* *equivalence) as long as the observed data has a probability of at least $1 - \alpha$. Refer to standard statistical texts, such as [63] to choose an appropriate value for α.*

3.2.2 Measuring Distance

When two models are not equivalent, the distance between them is determined by how much their statistics differ with significance α[1]. To find this difference, we progressively remove the least likely events from both systems, until the remaining Markov processes are equivalent according to Algorithm 3.2.1. Note that this is fundamentally different from finding graph isomorphism or the maximum common subgraph (MCS) problem from graph theory. Instead, this finds the largest common sub-models of G_1 and G_2 that represent the same underlying process. In the rest of this section, we discuss how to find the least likely events in HMMs. A threshold is then set to remove events in the models until there is an equivalence.

To find the least likely events, we use joint probability calculation from [97]. Let event X be the transition from state v_i to state v_j. The asymptotic probability of X is

$$P(X) = s_i p_{i,j} \tag{3.3}$$

where s_i is the asymptotic state probability for state $v_i \in V$ and $p_{i,j}$ is the probability of the outgoing transition to state $v_j \in V$. We use event probability $P(X)$ as a threshold value P_{th} and events in the model (process) with joint probability not greater than P_{th} are deleted.

The threshold value P_{th} then sets a minimum value, $\beta_i = P_{th}/s_i$ for each state with asymptotic state probability s_i. We then remove from each state v_i the outgoing transitions with probability $p_{i,j}$ not greater than β_i. After removing these transitions from the model, we remove transient and absorbing states from the new model, along with transitions going into or out of these states. Once low probability transitions and states are removed, the model is renormalized. For each remaining state, the probabilities of transitions leaving the state are rescaled to sum to 1. This process is shown in Figure 3.5 using a simple model with threshold value $P_{th} = 0.002$. This example uses P_{th} to remove events in one model. While comparing two models, P_{th} are used to prune events in both models following the same process.

The absolute bounds on P_{th} are $[0, 1]$, since it is a probability. If P_{th} is set to 1, all transitions

[1]Different α values will have different metric values. This is a side-effect of using a statistical approach.

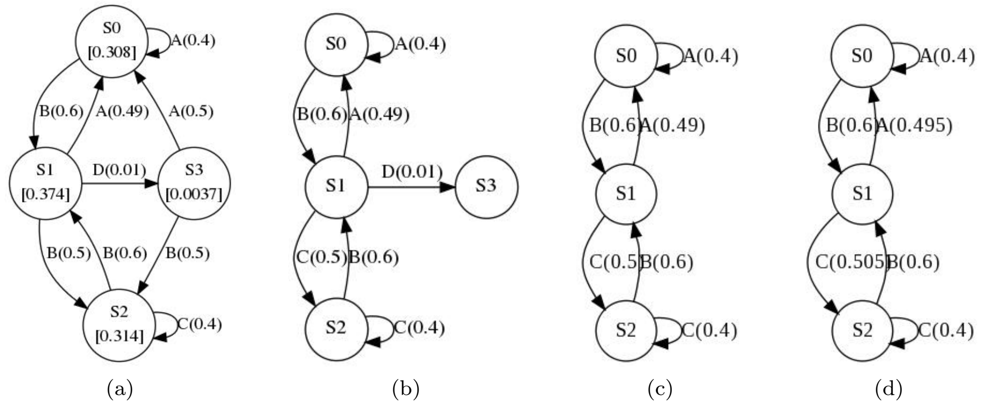


Figure 3.5: Process to remove transitions; (a) original model (asymptotic state probabilities are in square brackets); (b) initial step with $P_{th} = 0.002$; (c) removal of absorbing states; (d) resulting model with rescaled probabilities.

and states are removed. If P_{th} is set to 0, no states and transitions are removed from the models. From Equation (3.3) we see that as P_{th} increases, the minimum outgoing transition probability β_i increases, causing more transitions and states to be removed from the models. In Section 3.2.6, we discuss how to choose appropriate P_{th} values.

3.2.3 Metric Definition

We define our distance function $d(\cdot)$ to be the minimum value of P_{th} required to make two models equivalent. Two null models are equivalent. From this definition, we show first that $d(G_1, G_2)$ is a metric. A metric is a function $d : m \times m \to \mathbb{R}$, where m is a space (for us the space is all HMMs), that fulfills the following conditions [94]:

• (Non-negativity)	$d(G_1, G_2) \geq 0$
• (Self-equivalence)	$d(G_1, G_1) = 0$
• (Symmetry)	$d(G_1, G_2) = d(G_2, G_1)$
• (Identity of indiscernibles)	$d(G_1, G_2) = 0$ iff $G_1 \equiv G_2$
• (Triangle inequality)	$d(G_1, G_2) + d(G_2, G_3) \geq d(G_1, G_3)$

Proposition 3.2.1. *$d(G_1, G_2) = P_{th}$, where P_{th} is the smallest threshold probability needed to make $G_1 \equiv G_2$ according to Algorithm 3.2.1, is a metric.*

Proof. The proof for each property follows.

Non-negativity: $d(\cdot)$ is defined as a probability, which can never be less than zero.

Self-equivalence: Since $V_1 = V_1$ and $P_1 = P_1$, no transitions need to be removed in order for G_1 to accept sequences generated by itself. Zero is the minimum value for P_{th} that makes G_1 and G_1 equivalent.

Symmetry: By definition, $d(\cdot)$ is the minimum P_{th} needed to make the two models equivalent. This unique value will be the same value no matter which model is named G_1 or G_2.

Identity of indiscernibles: If $G_1 \equiv G_2$, there is no need to remove transitions from either model. The minimum P_{th} is 0, and $d(G_1, G_2) = 0$. Conversely, if $d(G_1, G_2) = 0$, by definition, the value for P_{th} is 0, which means no transition needs to be removed to make these two models equivalent. So $G_1 \equiv G_2$.

Triangle inequality: When $d(G_1, G_2) = p_1$, all events with joint probability not greater than p_1 in both models are removed creating models G_1' and G_2', where $G_1' \equiv G_2'$. At this point the two models are statistically equivalent. By Algorithm 3.2.1, for G_1 and G_2, all joint events with probability greater than p_1, corresponding to G_1' and G_2' have equivalent statistics. Similarly when $d(G_2, G_3) = p_2$, we get G_2'' and G_3'' with $G_2'' \equiv G_3''$. By Algorithm 3.2.1, for G_2 and G_3, all joint events with probability greater than p_2, corresponding to G_2'' and G_3'' have equivalent statistics.

Without loss of generality, assume $p_2 \geq p_1$. Notice that the difference between G_2', G_2'' are the events with joint probabilities in range $[p_1, p_2]$ in G_2. Since $G_1' \equiv G_2'$ and $G_2'' \equiv G_3''$, we can remove events with joint probabilities in $[p_1, p_2]$ in G_1 to make G_1'' which is equivalent to G_3''. This makes the distance $d(G_1, G_3)$ at most p_2, which proves the triangle inequality $d(G_1, G_3) \leq p_2 \leq p_1 + p_2 = d(G_1, G_2) + d(G_2, G_3)$. □

The distance function, $d(G_1, G_2)$, therefore satisfies all the conditions for being a metric. In Sections 3.2.4 and 3.2.9 we show how this metric can be used to compare HMMs.

3.2.4 Distance Calculation

In this section, we introduce the distance calculation algorithm using the Markov metric from the previous section. We first discuss the sample size needed by the HMM equivalence test. Then critical event probabilities are chosen as threshold values for distance calculation. At last, the distance calculation approach is introduced to find the minimum threshold value that makes two

HMMs equivalent.

3.2.5 Sample Size

Since Algorithm 3.2.1 uses the χ^2 test, we need to ensure enough samples are used for transition probabilities to adequately approximate normal distributions. Assume that a state is entered n_i times, then each outgoing transition can be represented with a binomial distribution where the expected number of times that the state is exited on a transition is $n_i p_{i,j}$, where $p_{i,j} \in P$ is the probability of the outgoing transition. A binomial distribution may be represented with a normal distribution if $p \pm 2\sqrt{p(1-p)/n} \in [0, 1]$, [72]. Therefore, the number of times n_i the state must be entered in order for all outgoing transitions to be represented by normal distributions is at least

$$n_i = \left\lceil \max\left\{ \frac{4p_{i,j}}{1-p_{i,j}}, \frac{4(1-p_{i,j})}{p_{i,j}} \right\} \right\rceil \quad \forall j \tag{3.4}$$

For a given state, n_i is the product of the asymptotic state probability and the length of the data sequence ($n_i = N_i s_i$). We find the corresponding asymptotic state probability s_i in the model. So the number of samples needed for the model is at least

$$N = \max\{N_i\} = \max\left\{ \left\lceil \frac{n_i}{s_i} \right\rceil \right\}, \forall i \tag{3.5}$$

Using sequences of length N or greater ensures that all transitions in the model can be represented with normal distributions and transitions are taken enough times in the model for the χ^2 test to have a sufficient number of samples. More samples will provide a more accurate result. This explains why we use the HMM with the fewest degrees of freedom to test the sequence: each transition is taken more times giving a more accurate result.

3.2.6 Threshold Values

We use event probabilities as threshold values for removing transitions and states. The probabilities of some events may not be significantly different, limiting our ability to reliably differentiate between them. For models derived from observed data, these differences are likely to be data sampling artifacts. When removing one event, events that are not significantly different should also be removed. We use Equation 3.6 from [95] to determine if two probabilities are significantly different:

$$\lambda = \log\left(\frac{q_1(1-q_2)}{(1-q_1)q_2}\right) \tag{3.6}$$

where log is the natural logarithm, and q_1, q_2 are probabilities. According to [95], if $|\lambda| < 0.41$, two probabilities can be accepted as equivalent, otherwise we reject the hypothesis and treat them as distinct. Events whose probabilities do not differ significantly are grouped together. The largest probability of each group is used for the P_{th} value. Algorithm 3.2.2 describes this procedure.

Algorithm Description 3.2.2 – Threshold Values

```
Input: HMM $G_1$; HMM $G_2$;
1. Calculate the joint probabilities $P(X)$ of events in both models, sort them from high to low and store
   them in an array $q[i]$;
2. Push 1 and the highest event probability $q[0]$ into a stack $S$;
3. Test $q[i]$ from high to low:
   (a) If $q[i]$ is equivalent with the top probability in stack $S$ by Equation 3.6, continue;
   (b) Otherwise push $q[i]$ into stack $S$;
   (c) Increment $i$;
4. Push number 0 into stack $S$ as the lowest value of $P_{th}$.
5. Return probabilities in stack $S$ as critical values of $P_{th}$.
```

Note that $P_{th} = 0$ is used to test the equivalence of original models without removing any transitions and states. $P_{th} = 1$ is used to prune all the events in both models.

3.2.7 Distance Calculation

From the metric definition, we find the minimum P_{th} that makes two models equivalent. So P_{th} iterates from low to high and stops when two models are equivalent. This P_{th} value is the distance between two models. Algorithm 3.2.3 describes the distance calculation for HMMs.

The computational complexity of this algorithm is $O(Nn_sn_t)$, where $O(Nn_s)$ is the complexity to check model equivalence and n_t is the number of transitions, since all transitions might be removed to calculate the distance. In our implementation, 0 and 1 are always included in the P_{th} value set, so an answer will always be returned upon exit. The minimum distance of 0 means that two models are equivalent, and the maximum distance of 1 means that they are totally different.

Algorithm Description 3.2.3 – Distance Calculation

```
Input: HMM $G_1$; HMM $G_2$; Statistical Significance $\alpha$
1. Determine the threshold values $P_{th}$ from event probabilities;
2. Iterate through $P_{th}$ values from lowest to highest:
    (a) Remove events whose probability is not greater than $P_{th}$ from both models;
    (b) Renormalize transition probabilities to get $G'_1$ and $G'_2$;
    (c) If either $G'_1$ or $G'_2$ is a null model, continue to the next $P_{th}$ value;
    (d) Else determine the sequence length $N$ from Equation 3.4 and 3.5 for $G'_1$ and $G'_2$;
    (e) Use Algorithm 3.2.1 with $\alpha$, $N$, $G'_1$ and $G'_2$ to determine if $G'_1 \equiv G'_2$;
    (f) If there is an equivalence or the highest value is reached, exit; else, continue to the next $P_{th}$
        value;
3. The value of $P_{th}$ upon exit is the distance between $G_1$ and $G_2$.
```

3.2.8 Comparison to KLD

We note that our approach is similar to the Kullback-Leibler divergence (KLD) approach for comparing HMMs [48], [81]. KLD measures the distance between two continuous probability density functions (PDFs). Since HMMs have discrete PDFs, there is no closed form of KLD for comparing HMMs [48]. A Monte-Carlo approximation is usually used to measure the entropy divergence between two models (G_1, G_2) with a sequence of observations [48]:

$$D(G_1, G_2) = \frac{1}{T}[logP(O_T|G_1) - logP(O_T|G_2)] \tag{3.7}$$

where O_T is an observation sequence generated by G_1 and T is the sequence length. This measure is not symmetric, but can be made symmetric [70] as:

$$D_s(G_1, G_2) = \frac{1}{2}[D(G_1, G_2) + D(G_2, G_1)] \tag{3.8}$$

Though this approach is widely used, it has several limitations. First, it does not satisfy the properties of a metric, such as the triangle inequality [99]. Second, it only works when both probabilities exist. A noise event in one model may cause the other model to reject the sequence, resulting in a non-computable probability. As a result, KLD can not measure the distance between some models even though they may be identical except for a small amount of noise. Third, the KLD result depends on the sample size[2]. If the sample size is too large, the computational cost is high and

[2]Detailed discussion about the accuracy of Monte Carlo method can be found in [34]

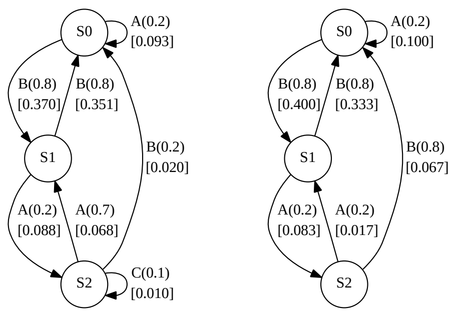
the calculated probabilities may suffer from floating point underflow. If the sample size is too small, the result may be unstable [71] [99]. These limitations exist in variants of this approach, including those in [81, 100, 99, 32]. However, our approach does not suffer from these problems. It provides a normalized metric similar in spirit to the Levenshtein metric on strings [62]. By systematically removing low probability events, at a certain point, we remove noise events and return an appropriate value if the models are otherwise equivalent. Also we can automatically calculate the sample size necessary for the statistical test. The χ^2 test is more appropriate than KLD for comparing discrete PDFs like those in a HMM. The test also determines the statistical confidence of the metric value. Therefore our approach is more appropriate and effective for comparing HMMs.

3.2.9 Verification

In this section, we present an example to demonstrate the above approaches. To illustrate the performance of the Markov metric, we calculate distances between different, similar, and equivalent models. In all examples, we use $\alpha = 0.05$. We compare the KLD results with our approach to show that, in these cases, our approach outperforms KLD, because our metric always returns an appropriate distance measure.

3.2.9.1 Illustrative Example

We show how to calculate the distance between models G_1 and G_2 in Figure 3.6. G_1 has an additional self-loop and different transition probabilities leaving state $S2$. The event probabilities (by Equation 3.3) are shown as square brackets in the figure. The threshold values calculated using Algorithm 3.2.2 are 0, 0.010, 0.020, 0.068, 0.100, 0.400 and 1.



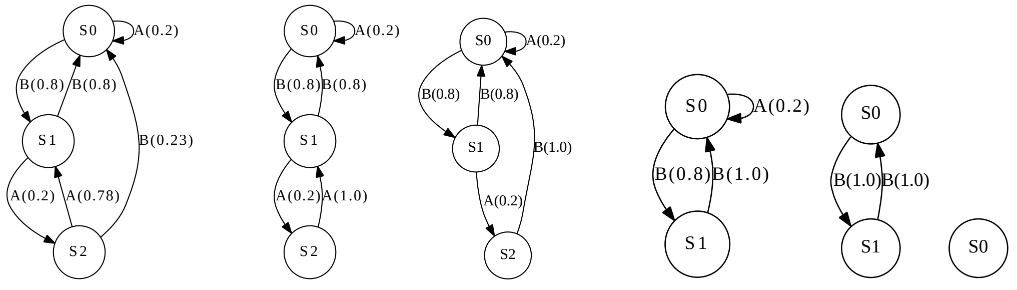
(a) G_1 (b) G_2

Figure 3.6: Designed models to illustrate the algorithm. (Event probabilities are in square brackets)

Intermediate results of Algorithm 3.2.3 are given in Table 3.1. Each row shows the result for its respective P_{th} value. Columns marked with G_1' and G_2' are remaining models (shown in Figure 3.7) of G_1 and G_2 respectively after pruning events below P_{th}. N is the lower bound sample size needed for equivalence test from Equation 3.5. Algorithm 3.2.1 results are the last column of the table.

Table 3.1: Distance calculation results

P_{th}	G_1'	G_2'	N	Test result
0	G_1	G_2	369	not equivalent
0.010	G_3	G_2	204	not equivalent
0.020	G_4	G_5	122	not equivalent
0.068	G_6	G_6	30	equivalent
0.100	G_7	G_7	-	equivalent
0.400	G_8	G_8	-	equivalent
1	G_8	G_8	-	equivalent



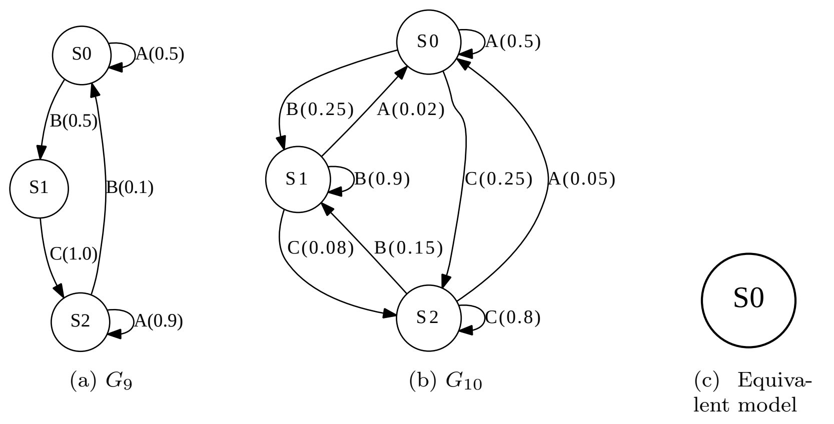
(a) G_3 (b) G_4 (c) G_5 (d) G_6 (e) G_7 (f) G_8

Figure 3.7: Remaining models for G_1 and G_2 in Figure 3.6 with different P_{th} values.

For $P_{th} = 0$, Algorithm 3.2.1 says the models differ. At $P_{th} = 0.010$, G_1 transforms to model G_3 in Figure 3.7a and G_2 remains the same. Algorithm 3.2.1 correctly rejects the equivalence even though they have same structure due to the transition probabilities leaving state $S2$. Continuing the threshold value tests, equivalence is reached when $P_{th} = 0.068$. At this point, G_1 and G_2 are both reduced to model G_6 in Figure 3.7d. Therefore the distance between G_1 and G_2 is 0.068. The equivalent portion of G_1 and G_2 is model G_6. Once the 0.068 threshold value is reached and equivalence is shown, further threshold values will continue to hold equivalence. For illustrative purposes the entire P_{th} set is provided in Table 3.1.

We next discuss the performance of the metric when applied to different, similar, and equivalent HMMs, using the same procedures described above. Symmetric KLD measurement is also applied to each example for comparison. The observation sequences of length $T = 1000$ is used, since larger T may cause floating point underflow.

3.2.9.2 Different models



(a) G_9 (b) G_{10} (c) Equivalent model

Figure 3.8: Different models

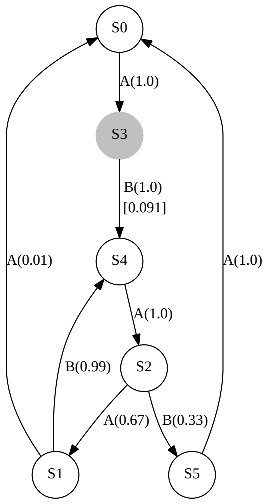
We calculate the distance between two different models G_9 and G_{10} in Figure 3.8 by Algorithm 3.2.3. A cursory look at the models clearly shows that both the state structure and underlying process are completely different. This is confirmed by Algorithm 3.2.3 showing equivalence only when $P_{th} = 1$, which means that they are totally different models. The equivalent portion of the models is a single state, shown in Figure 3.8c.

The KLD does not exist for G_9 and G_{10}, since each model can not accept the sequence generated by the other model. So $P(O_{T_j}|G_i) = 0$ and $D(G_i, G_j) = \infty$.

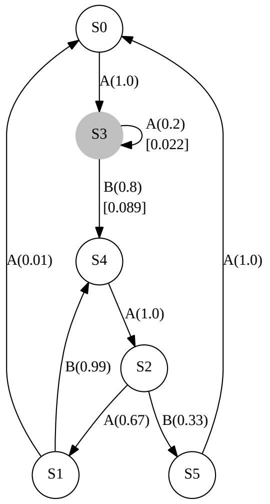
3.2.9.3 Similar models

Algorithm 3.2.3 is used to calculate the distance between G_{11}, G_{12} and G_{13} in Figure 3.9. They have similar structures and are only different in the transitions around state $S3$. Event probabilities of transitions leaving state $S3$ (where the differences are located) are in square brackets. The results are in Table 3.2.

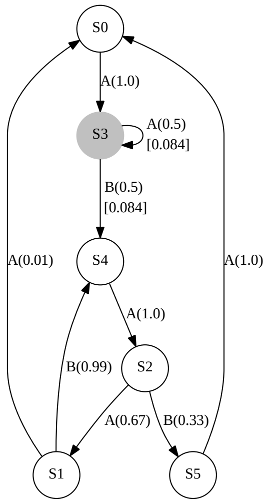
The difference between G_{11} and G_{12} is the transition $\delta(S3, S3) = A$ in model G_{12}. After removing this event, which has a joint probability of 0.022, G_{11} and G_{12} become equivalent. For model G_{11} and G_{13}, since the probabilities of transition $\delta(S3, S4) = B$ in G_{11} and transitions



(a) G_{11}



(b) G_{12}



(c) G_{13}

Figure 3.9: Similar models

Table 3.2: Results for similar models

Test pair	Distance
$G_{11} - G_{12}$	0.022
$G_{11} - G_{13}$	0.091
$G_{12} - G_{13}$	0.089

$\delta(S3, S3) = A$ and $\delta(S3, S4) = B$ in G_{13} are not significantly different, they are grouped and we use the largest event probability (0.091) for P_{th}. G_{11} and G_{13} become equivalent after removing events below 0.091, resulting in our distance value. This is larger than the distance between G_{11} and G_{12} since the difference in transition probabilities is larger. Following the same procedure, the distance between models G_{12} and G_{13} is 0.089 since the differences between the models are two transition probabilities. A smaller distance indicates that the models are more similar. With this example, we also provide an illustration of the triangle inequality.

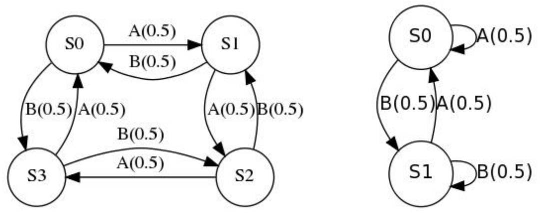
KLD is also calculated for these models. The results are in Table 3.3. Since G_{11} can not accept the sequence generated by G_{12} due to the additional transition in $S3$, $D(G_{12}, G_{11}) = \infty$. However, $D(G_{11}, G_{12})$ is small since G_{12} can match the sequence generated by G_{11} with a similar probability. It is same for G_{11} and G_{13}. Both cases clearly show that KLD can not measure the distance for them, however, our approach solves this problem by progressively removing events to a point where the remaining models are equivalent, and returns a reasonable distance value. This shows a strong point of our approach in that it can measure the distance between all appropriate models, while maintaining the triangle inequality.

Table 3.3: KLD results for different models

Test pair	$D(G_i, G_j)$	$D(G_j, G_i)$	$D_s(G_i, G_j)$
$i = 11, j = 12$	0.008	∞	∞
$i = 11, j = 13$	0.028	∞	∞
$i = 12, j = 13$	0.012	0.016	0.014

3.2.9.4 Equivalent models

HMMs in Figure 3.10 are statistically equivalent since for both models, every state has the same probability of generating symbols A and B, and every state is equally likely to be visited. So the distance between the models should be 0. If Algorithm 3.2.3 says the distance is not 0, we count that as a false positive. We vary sample size N from 4 to 500 to see the effect of N in equivalence test. For each N, we repeat the distance calculation 1000 times. The results are provided in Table 3.4.



(a) G_{14} (b) G_{15}

Figure 3.10: Equivalent models.

Table 3.4: False positive rate out of 1000 times with different sample sizes

Sample size N	4	8	12	[16]	25	50	100	200	350	500
FP rate	0%	1.3%	2.4%	4.0%	4.4%	5.3%	5.5%	4.9%	4.8%	5.1%

The first row of Table 3.4 lists different sample sizes of N. $N = 16$ is the minimum sample size needed for the normal distributions to be constructed and the equivalence test from Equation 3.5 to be valid. The second row shows the false positive rates with different N. For $N \geq 16$, the false positive rate converges to the statistical significance level α. We expect this because α is the probability of incorrectly rejecting the null hypothesis in the statistical test and was previously set to 0.05 for all tests. For $N < 16$, it results in fewer effective false positives. This occurs because the confidence interval size is too large to detect deviations when the sample size is too small for normal distributions to be constructed. To get an accurate result, it is necessary to collect more samples than the minimum sample size calculated by Equation 3.5. In this case, KLD can measure the distance between them since each can match the sequence generated by the other.

From these examples, we can see that our metric is appropriate for comparing HMMs. Compared to KLD, our metric is a true metric and always returns a valid distance value.

3.3 Detection With PCFGs

Since HMMs are probabilistic regular grammars, the simplest class in Chomsky Hierarchy of grammars, they can only represent patterns in regular expressions, and can not detect recursive patterns [64]. Therefore it is interesting to apply context-free grammars (CFG) to recognize and detect recursive patterns when the stochastic pattern includes recursions.

Probabilistic (stochastic) CFGs (PCFGs) are CFGs where each production rule has an associated a probability. They can accommodate noise and pattern distortion, incorporate not only syntactic but also statistical information in data, and distinguish more plausible results from less plausible ones [58, 72]. PCFGs have been successfully applied to natural language processing [37, 58, 72], pattern analysis in RNA and protein sequences [31, 89, 90], and network data modeling [39]. In this work, we use PCFGs for pattern detection. For PCFGs, the ability to accurately detect rule probability is more important than to determine language membership.[3] In this paper, we assume production rules of the underlying system are known and data sets are languages of the grammar. Also, we only consider finite data sets.

Previous research on PCFGs, [89, 14, 47], usually uses the inside-outside algorithm [51, 52], which is a generalization of the Viterbi Algorithm for HMMs [70], for parameter estimation and data classification. For data classification purposes, the inside-outside algorithm finds the PCFG that most likely generated the data set, from several candidate PCFGs. The inside-outside algorithm is only useful for choosing between PCFGs. It can not detect if a data set fits a given PCFG. Furthermore, with large sample sizes, the inside-outside algorithm suffers from floating point underflow, similar to the Viterbi Algorithm [22]. It also has high computational complexity [58]. In this paper, we propose a simple statistical method for PCFGs to solve the following detection

[3]In fact, ill-formed strings can be modeled by production rules with extremely low probability. So the language membership problems can still be transferred into detecting rule probability problem.

problems:

1. Does an observed data set fit a given PCFG ?
2. Are two observed data sets generated by the same source PCFG ?

From data sets, we estimate production probabilities using a maximum-likelihood (ML) method. For tree data sets, it is easy to count the frequencies of nonterminals and productions in the tree. Production probability is estimated by relative frequency. We extend the ML estimation to sentence data sets. With known production rules, we build a LALR parser for the grammar to parse the sentence. While parsing, the frequency of use of each nonterminal and production is recorded. The production probability then can be estimated from the relative frequency. The ML estimation assigns proper and consistent PCFG distributions on finite data sets [25] [24].

With estimated PCFGs, we design a χ^2 test for pattern detection. A χ^2 test can be used for goodness-of-fit test. With the χ^2 test, we are able to tell if a data set matches a given PCFG, or another data set. Statistical tests require the parameter δ[4], which explicitly states the significance for comparing distributions [63]. With larger sample sizes, the χ^2 test has greater statistical power and higher confidence on the result. We provide illustrative examples to show that our approach has better detection rate than the inside-outside algorithm.

3.3.1 Production Probability Estimation

We estimate production probabilities from tree or sentence sets by Maximum Likelihood (ML) estimation [25, 24, 27]. Other approaches, such as the Expectation Maximization and Minimizing Cross Entropy (MCE) method[5] give the equivalent estimator for production probabilities [27]. PCFGs estimated by ML method are proper and consistent [25, 24]. In this section, we introduce the ML estimation for both tree sets and sentence sets.

3.3.2 Tree Sets

Given a finite trees set T, we use $f(A \rightarrow \alpha, t)$ to represent the frequency of the production $A \rightarrow \alpha$ occurs in a tree $t \in T$. Similarly, $f(A, t)$ represents the frequency of A occurs in t. The ML

[4]We use δ in stead of the usually used α because α is used to represented $(N \cup \Sigma)^*$ in PCFGs.

[5]It minimizes the cross entropy (or relative entropy) between the tree distribution induced by tree set and the tree distribution induced by the grammar.

estimates production probabilities from a tree set by equation:

$$\hat{p}(A \to \alpha) = \frac{\sum_{t \in T} f(A \to \alpha, t)}{\sum_{t \in T} f(A, t)} \tag{3.9}$$

It is the ratio between the number of occurrences of the production $A \to \alpha$ and the number of occurrences of the nonterminal A in the tree set. So the ML estimation is also called relative frequency estimation. It is easy to find that $f(A,t) = \sum_{\alpha} f(A \to \alpha, t)$. So after finding frequencies of all productions of a tree set, we can use Equation 3.9 to estimate a consistent PCFG.

3.3.3 Sentence Sets

For a sentences set S, we need a parser to parse the sentence to obtain the frequency of each production that is used while generating the sentence. Given production rules, we construct a common and efficient parser, LALR (LookAhead-Left-to-Right) parser [13]. We choose LALR parsers since their parsing table is considerably smaller than other LR parsing tables, and most common syntactic structures can be expressed conveniently[13]. In the following section, we introduce the derivation of a LALR parser from production rules and ML estimation for sentences.

3.3.3.1 LALR parsing table construction

A LALR parsing table is in a finite-state machine format. A LALR parsing table contains an action table and a goto table. The action table is indexed by a parser state and a terminal, including a special terminal $ that means the end of the sentence. There are 3 actions in the action table: *"shift"*, *"reduce"* and *"accept"*. Shift, written *"sn"*, indicates the parser will *"shift"* to the next input terminal and go to the next state *"n"*. Reduce, written as *"rm"*, indicates the parser will *"reduce"* the production rule *"m"*. Accept, written as *"acc"*, means the parser *"accept"*s the input sentence. The *"goto"* table is indexed by a parser state and a nonterminal. It shows the next state the parser goes to after each reduce action is performed.

For most CFGs, there exists a straight forward but space-consuming LALR table construction method [13], described in Appendix B. There are also other approaches to construct LALR parsing tables [13]. In general, LALR parsers are difficult for humans to construct [13]. So usually a LALR parser generator is used to create the parser table automatically from a grammar. In this paper, we use the beaver [1] to generate LALR parsers.

Table 3.6 shows the LALR parsing table of the CFG example in Chapter 2, shown in Table 3.5 as well. In Table 3.6, "s#" means to shift to state #, and "r#"means reduce the $\#^{th}$ production of the CFG.

Table 3.5: CFG Example

Terminals	a, b
Nonterminals	S, C
Productions	$S \rightarrow CC$ $C \rightarrow aC$ $C \rightarrow b$

Table 3.6: LALR parsing table

STATE	ACTION			GOTO	
	a	b	\$	S	C
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5			r1		
6	r2	r2	r2		

3.3.3.2 Estimation by parsing

Once the LALR parsing table is constructed, we use the extended ML method for trees to estimate production probabilities for sentences. Algorithm 3.3.1 describes this method.

Note that $n_i = \sum_j n_{i,j}$, therefore the estimated PCFG is proper. Table 3.7 gives a simple example of the production probability estimation by parsing a sentence "bab", which is generated by the derivation of $S \Rightarrow CC \Rightarrow bC \Rightarrow baC \Rightarrow bab$.

For this example, $\hat{p}(C \rightarrow aC) = \frac{n_{2,1}}{n_2} = \frac{1}{3}$, and $\hat{p}(C \rightarrow b) = \frac{n_{2,2}}{n_2} = \frac{2}{3}$. In general, if we use $f(r(A \rightarrow \alpha), s)$ to denotes the number of the reduce action of production $A \rightarrow \alpha$ used while parsing the sentence $s \in S$, and use $f(A, s)$ to denote the number of nonterminal A used in parsing, the ML estimation for probability of production $A \rightarrow \alpha$ is:

$$\hat{p}(A \rightarrow \alpha) = \frac{\sum_{s \in S} f(r(A \rightarrow \alpha), s)}{\sum_{s \in S} f(r(A), s)} \tag{3.10}$$

Since parsing is the process of reconstruction the tree structure from a sentence, ML esti-

Algorithm Description 3.3.1 – Estimation by parsing

```
Input: An input sentence; an LALR parsing table;
1. Push state 0 into the state stack $T$, initialize the grammar symbol stack $Y$ and start from the first terminal in the input sentence;
2. Set a counter $n_i$ for $i$th nonterminal, say $A$, and a counter $n_{i,j}$ for $j$th production rule that starts with $A$, say $A \rightarrow \alpha$;
3. While parsing is not finished, find the action determined by the current state (top state in $T$) with the current input terminal of the input sentence from the action table:
    (a) If the action is "shift $x$", push the state $x$ into $T$, push the current symbol into $Y$, and shift the current input terminal forward to the next input terminal;
    (b) If the action is "reduce $A \rightarrow \alpha$", pop out $k$ elements out of $T$ and $Y$, where $k$ is the number of grammar symbols in $\alpha$. Push $A$ into $Y$. Increment $n_{i,j}$ and $n_i$ corresponding to $A$. After reduction, find "GOTO $x$" action determined by the current state and the top nonterminal of $Y$, and push the state $x$ into $T$;
    (c) If the action is "accept", finish parsing.
4. After parsing, $n_i$ records the frequency of $A$ and $n_{i,j}$ records the frequency of production $A \rightarrow \alpha$;
5. The estimated probability of production $A \rightarrow \alpha$ is $\hat{p}_{i,j} = \frac{n_{i,j}}{n_i}$.
```

Table 3.7: An example of estimation by parsing

Step	T	Y	Input	Action	Counter Update
(0)	0		bab$	s4, (shift to state 4)	
(1)	0 4	b	ab$	r3, (reduce $C \rightarrow b$)	$n_{2,2} = 1$, $n_2 = 1$
(2)	0	C	ab$	s3	
(3)	0 3	Ca	b$	s4	
(4)	0 3 4	Cab	$	r3 (reduce $C \rightarrow b$)	$n_{2,2} = 2$, $n_2 = 2$
(5)	0 3	CaC	$	GOTO6	
(6)	0 3 6	CaC	$	r2 (reduce $C \rightarrow aC$)	$n_{2,1} = 1$, $n_2 = 3$
(7)	0	CC	$	GOTO2	
(8)	0 2	CC	$	GOTO5	
(9)	0 2 5	CC	$	r1 (reduce $S \rightarrow CC$)	$n_{1,1} = 1$, $n_1 = 1$
(10)	0	S	$	GOTO1	
(11)	0 1	S	$	accept	

mation for a sentence set by parsing is equivalent with ML estimation for a tree set. Therefore, we just consider tree sets in the following discussion without loss of generality.

3.3.4 PCFG Detection

In this section, statistics are used to detect recursive patterns. We use a χ^2 test to detect whether or not two data sets match (generated by the same source PCFG), and whether or not a data set matches a PCFG (meaning the data set is generated by this PCFG). χ^2 tests test whether or not two probability distributions are equal [91, 101]. For a χ^2 test, a statistical significance level δ

is necessary, which defines the false positive rate the user can tolerate. It also provides a confidence on the detection result. Refer to standard statistical texts, such as [63] to choose an appropriate value for δ.

To match a tree set T and a known PCFG, we define the statistic:

$$\chi^2 = \sum_i \sum_j \frac{(n_{i,j} - n_i p_{i,j})^2}{n_i p_{i,j}} \tag{3.11}$$

where $n_i = f(A, T)$, $n_{i,j} = f(A \rightarrow \alpha, T)$ and $p_{i,j}$ is the probability of the same production $A \rightarrow \alpha$ in the known PCFG.

The degrees of freedom (DOF) of the test is $|R| - |N|$, where $|R|$ is the number of production rules and $|N|$ the number of nonterminals. With statistical significance δ and the DOF, we find the critical value χ^2_δ in a standard statistics table. If $\chi^2 < \chi^2_\delta$, we accept the hypothesis that T matches with the PCFG with a significance level of δ, otherwise we reject the hypothesis.

To determine if tree set T_1 and tree set T_2 are equivalent, we use a different statistic:

$$\chi^2 = \sum_i \sum_j \frac{(n_{i,j} - n_i \tilde{p}_{i,j})^2}{n_i \tilde{p}_{i,j}} + \sum_i \sum_j \frac{(m_{i,j} - m_i \tilde{p}_{i,j})^2}{m_i \tilde{p}_{i,j}} \tag{3.12}$$

where $n_i = f(A, T_1)$ is the number of a nonterminal A in T_1, $n_{i,j} = f(A \rightarrow \alpha, T_1)$ is the number of production $A \rightarrow \alpha$ in T_1, $m_i = f(A, T_2)$, $m_{i,j} = f(A \rightarrow \alpha, T_2)$, and $\tilde{p}_{i,j} = \frac{n_{i,j}+m_{i,j}}{n_i+m_i}$. Similarly, if $\chi^2 < \chi^2_\delta$, we accept that they are equivalent, otherwise they are not equivalent.

3.3.5 Illustrative example

To show the PCFG detection approach, we give a simple example using the parentheses matching CFG with 10 different sets of production probabilities, shown in Table 3.8. Since we only consider finite trees, we choose production probabilities so that the branching rate of the PCFG is less than 1. In this example, the branching rate[6] is $\rho = P[A \rightarrow (A)] + 2P[A \rightarrow AA]$. So we vary three production probabilities to satisfy the condition $\rho < 1$ and their sum to be 1.

From each of these ten PCFGs, we generate 50 sets, and each contains 30 sentences (around 100 to 400 terminals). We first test the equivalence between a set of sentences and a PCFG. If the set is generated by the PCFG, and it is accept that they are equivalent, we count it as a true positive

[6]The calculation of branching rate refers to [24]

Table 3.8: PCFGs for parentheses matching

Production rules	PCFG1	PCFG2	PCFG3	PCFG4	PCFG5
$A \to (A)$	0.1	0.1	0.2	0.2	0.3
$A \to AA$	0.1	0.2	0.1	0.2	0.1
$A \to ()$	0.8	0.7	0.7	0.6	0.6
Production rules	PCFG6	PCFG7	PCFG8	PCFG9	PCFG10
$A \to AA$	0.3	0.4	0.4	0.5	0.5
$A \to AA$	0.2	0.1	0.2	0.1	0.2
$A \to ()$	0.5	0.5	0.4	0.4	0.3

(TP). If the set is generated by a different PCFG, but it is detected as equivalent, we have a false positive (FP). And we use $\delta = 0.05$ as the significance level.

Table 3.9 shows the detection result. Our approach has low false positive rates and true positive rates are close to the pre-defined TP rate $1 - \delta = 0.95$. The receiver operating characteristic (ROC) curve is used to evaluate the detection rate of our approach, since the ROC curve consider both TP rate and FP rate. The last column in Table 3.9 shows the Euclidean distances between each PCFG detection rate and the optimal point (0,1), where there is no false positives and 100% true positives.

Table 3.9: χ^2 detection between a data set and a PCFG

PCFGs	TP rate	FP rate	Distance
PCFG1	0.96	0.16	0.165
PCFG2	0.94	0.21	0.218
PCFG3	0.98	0.27	0.274
PCFG4	0.94	0.29	0.303
PCFG5	0.96	0.28	0.283
PCFG6	0.92	0.22	0.238
PCFG7	0.90	0.18	0.206
PCFG8	0.98	0.12	0.122
PCFG9	0.92	0.08	0.112
PCFG10	0.92	0.03	0.085

We compare the performance of our approach with the inside-outside method based on the same data sets. For each data set S, the algorithm calculates the probability $P(S|G_i)$ that this data set is generated by a PCFG G_i, by recursively multiplying the probability of each production is used while parsing the data. After calculating $P(S|G_i)$ for each G_i, it then selects the largest probability $P(S|G_i)$, and corresponding G_i is the PCFG that best fit the data set. For each data set, this algorithm only returns one (and exactly one) matched PCFG, while for our approach, it may find no model or more than one model that matches the data. This is also the difference between detection and classification.

Table 3.10: ML detection between a data set and a PCFG

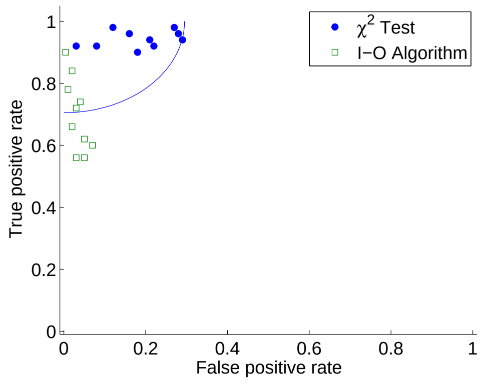
PCFGs	TP rate	FP rate	Distance
PCFG1	0.62	0.05	0.383
PCFG2	0.72	0.03	0.281
PCFG3	0.6	0.07	0.407
PCFG4	0.66	0.02	0.341
PCFG5	0.56	0.05	0.442
PCFG6	0.74	0.04	0.263
PCFG7	0.56	0.03	0.441
PCFG8	0.78	0.01	0.220
PCFG9	0.84	0.02	0.161
PCFG10	0.9	0.004	0.100

The detection result of the inside-outside algorithm is shown in Table 3.10. It has lower FP rates, but TP rates are also much lower than our approach. Also, the distance between between the detection point and (0,1) is larger, which means our approach have better performance in terms of high TP rates as well as low FP rates.

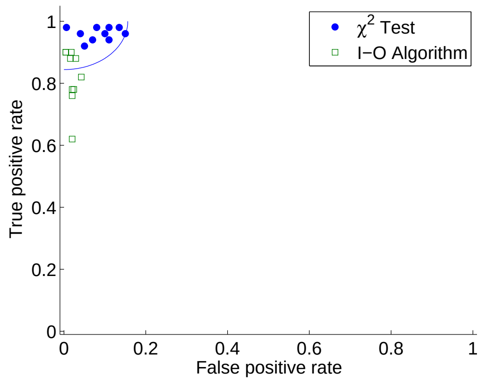
The sample size is important in the test. If the sample size is too small, both detection approaches may not be accurate. If the sample size is large, the χ^2 detection will have lower FP rate and higher confidence on the result; while for the inside-outside algorithm, it may suffer floating point underflow. We repeat the detection process using data sets with more samples. For each PCFG, we generate 50 data sets and each contains 50 sentences (around 200 to 1,500 terminals). Similarly, we can get the detection result using both approaches.

Figure 3.11 shows the detection rates for our χ^2 approach and the inside-outside (I-O) algorithm with different sample sizes. It is clear that with larger sentence sets, false positive rates of our approach get lower. By comparing the distance between the detection point to the optimal point (0,1), our approach still outperforms the inside-outside algorithm. For data sets containing more than 50 sentences, a lot of probabilities calculated by the inside-outside algorithm have floating point underflow, and therefore this approach has worse detection rates. The χ^2 approach, however, maintains high TP rates and even lower FP rates.

From this example we can see that our approach has better detection rates than the inside-outside algorithm. Our approach considers the number of data samples available, and increases the confidence on the detection result by reducing FP rates with larger data samples, while the inside-outside algorithm may encounter floating point underflow problems.



(a) Each data set containing 30 sentences



(b) Each data set containing 50 sentences

Figure 3.11: Detection rate with different sample sizes.

Our approach can also detect if two data sets are generated from the same source PCFG. This is an another advantage of our approach, because the inside-outside algorithm does not provide the criteria for determining if two data sets are consistent in probability distributions. Using larger data sets that we generated above, we show the performance of this detection approach. If two sets are generated by different PCFGs, but it is detected as equivalent, we have a false positive. If two sets are generated by a same PCFG, and it is accept that they are equivalent, we count it as a true positive. The detection result is shown in Table 3.11. It has a similar performance as the detection between a data set and a PCFG.

Table 3.11: Detection result between two data sets

Data source	PCFG1	PCFG2	PCFG3	PCFG4	PCFG5
TP rate	0.9288	0.9624	0.9504	0.9744	0.9664
FP rate	0.2108	0.1896	0.3155	0.2838	0.3091
Data source	PCFG6	PCFG7	PCFG8	PCFG9	PCFG10
TP rate	0.9632	0.9296	0.9728	0.9688	0.9528
FP rate	0.2516	0.2494	0.1491	0.1316	0.0439

Note in our detection examples, the FP rates are slightly high. This is because our test data sets are generated from PCFGs with quite similar probability distributions. It is more difficult to differentiate them when they are generated from subtly different processes. Generally more samples are required for the χ^2 test to distinguish two subtly different distributions.

3.4 Summary

In this chapter, we give detailed description about our detection approaches using HMMs and PCFGs. From a sequence of observations, we infer HMMs without *a priori* information. With the inferred HMMs, we apply the extended confidence intervals approach to detect if a new observed sequence matches the HMM.

To compare the similarity of HMMs, we propose a Markov metric. A statistical test is used to check the equivalence between HMMs, which focuses on the stochastic processes the HMM represents. With data sampling insights, we systematically remove states and transitions from HMMs to find a structure close to both models being compared, which expresses the portion of both models that are statistically equivalent. This structure then defines the distance between two models. Illustrative examples show that our metric provides appropriate distance values for HMMs and outperforms KLD method, because our metric is a true metric and always returns an appropriate distance measure.

To detect recursive patterns, we use PCFGs. We estimate production probabilities from observed data (both tree sets and sentences sets with the help of LALR parser) using a ML method. To determine whether a new data set matches a given PCFG or another data set, we use χ^2 tests for detection. With some illustrative examples, we show the performance of our detection approaches. Our approach can work on detection problems that the inside-outside algorithm can not solve, and has better detection rates in terms of high true positive rates and low false positive rates.

Chapter 4

Network Traffic Analysis

Using stochastic grammar detection approaches described in the previous chapter, we perform three applications exploiting traffic timing data. Since HMMs and PCFGs work with symbolized data, we first introduce the symbolization method used to translate inter-packet timings into symbols for learning stochastic grammars.

The first application is Tor (the onion router) network traffic analysis. We attempt to identify protocols (represented by HMMs) tunneled through Tor and break Tor's anonymity. However, noise in the Tor network obscures the underlying protocol. The Markov metric helps us identify the correct protocol used in Tor. The metric also measures the noise in the network.

We then detect centralized botnet traffic using a HMM. In this application, we focus on Zeus botnets (Zbots), one of the largest botnets. We infer HMMs from botnet traffic timing data and use inferred HMMs to detect botnet communication traffic. Experiment results show that we can adequately detect botnet traffic from normal traffic.

Centralized botnets, like Zbots, can be detected using HMMs [54, 55], and once one bot is detected, the centralized C&C server can be found and intercepted. To avoid this, hierarchical botnets use P2P techniques [5, 56, 66]. In the P2P botnet detection application, we first used HMMs detection approach to detect the traffic timing data from simulated hierarchical botnet in Clemson campus computer cloud (Palmetto cloud). The result shows that HMMs can not detect recursive patterns in P2P botnet traffic timings. This is because HMMs can only represent regular expressions. However, PCFGs can accurately detect the timing data of simulated hierarchical botnet, since these patterns are recursive.

Finally, we apply the PCFG detection approach to real-world traces of the Storm botnet traffic, which is one of the known P2P botnets [43]. We estimate the production probabilities from the Storm botnet traffic data and use the estimated PCFG for detection. The result shows that our approach can adequately distinguish botnet traffic from normal background traffic.

Timing profiles exist in all automated network processes, and encryption does not change traffic data timing. It is difficult to remove timing profiles without significant performance implications. Therefore, it is possible to extend these approaches to detect traffic data from other network processes as well.

4.1 Symbolization

As discussed in Chapter 1, we focus on inter-packet timings of network traffic data. Timing data can be easily obtained from traffic data, even when traffic is encrypted. Inter-packet timing provides information about the temporal properties of unknown underlying processes [59]. Furthermore, inter-packet timings filter out constant network delays, and timing signatures are preserved during network transmissions [92]. We use Tshark [7] to capture timings of packets. Inter-packet delays are calculated by finding time differences between successive packets of interest. In other words, $\Delta t_i = t_i - t_{i-1}$, where t_i is the receiving time of packet i.

Since our stochastic grammars work with symbolized data, we convert inter-packet delays into symbols. Similar delays are grouped together and replaced with a character. Different groups of delays can be found by plotting the data histogram or using clustering algorithms (k-means, self-organizing maps, etc). In this paper, we use local minima in the histogram of the delays to determine the timing ranges for packets. Figure 4.1 shows an example histogram of inter-packet delays. There are two distinct clusters so we use two symbols. More details about this approach can be found in [19, 18, 28, 12, 41]. With symbolic data, we are able to use them for our detection approaches using stochastic grammars.

4.2 TOR Network Traffic Analysis

In this section, we use HMM inference approach and Markov metric to analyze Tor network [6] traffic. A server is configured to communicate with a client through Tor based on given protocols

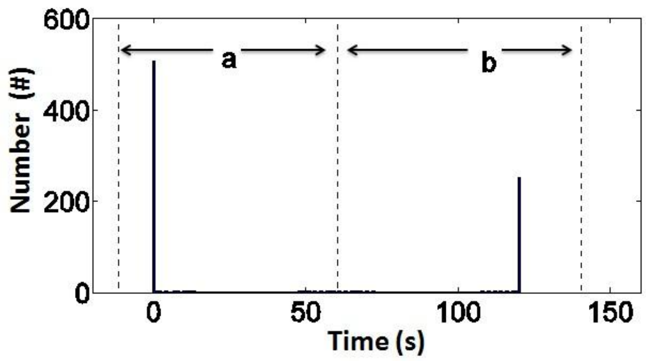


Figure 4.1: Histogram of inter-packet delays

(represented by HMMs). We attempt to infer HMMs from the traffic timing data. However, the inferred HMM has a lot of noise events and the underlying system is difficult to discern. Using the Markov metric, we identify the underlying model. Also, the Markov metric quantifies the noise in the Tor communication.

4.2.1 Application Setup

We created a small, standalone test network to conduct experiments on Tor, in order to not disturb the global Tor network on the Internet. The private Tor network consisted of sixteen nodes. All ran Tor version 0.2.1.19, a stable version released in August 2009 [6]. Two desktops ran CentOS 5, two other desktops ran Fedora Core 6, and twelve WebDT thin clients ran the lightweight Debian-based Damn Small Linux [2]. The network was a combination of different systems on different platforms, similar to the Internet.

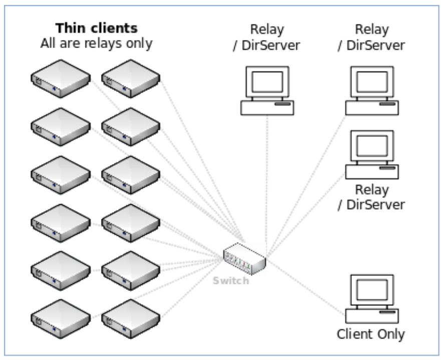


Figure 4.2: Network layout.

Figure 4.2 shows the network layout. The smaller icons on the left represent the thin clients. Each thin client ran Tor with the client and relay services enabled. Three of the desktops ran Tor with the client, relay, and directory server services enabled, and the last desktop ran Tor with just the client service enabled. All systems were connected via a network switch.

After building this private Tor network, we set up one server and one client. The client connected to the server through Tor and listened. The server sends packets based on a given HMM, which represents the protocol it uses. Figure 4.3 is the HMM used by the server. We use HMMs to represent network protocols because HMMs are finite state machines (FSMs), while network protocols can usually be represented by FSMs.

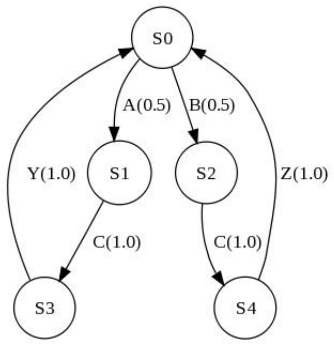


Figure 4.3: HMM M_1

To start, the server randomly selects a start state in the HMM, follows the transitions based on the transition probabilities, goes to another state, and generates an output symbol. This process is basically the same as generating a sequence from the HMM. Each output symbol is translated into a inter-packet delay. For each symbol, the server waited the specific amount of time defined by the symbol before sending the next packet to the client. Table 4.1 gives the symbol-delay translation we use in this application for the HMM M_1 in Figure 4.3.

Table 4.1: Symbol-delay translation table

Symbol	Delay time (in seconds)
A	0.2
B	0.4
C	0.1
Y	0.3
Z	0.5

We captured packet timings at the client with Tshark [7]. The timing difference Δt was calculated by subtracting the receive time of the previous packet from the time of the current packet. In a Tor connection, however, there are times when a circuit fails or changes, or a relay delays a packet, causing extra latency. Among other factors, this introduces noise into the communication. For symbolization, we used Table 4.2 as the delay-symbol lookup table by finding the local minima in the histogram of the delays.

Table 4.2: Delay-Symbol lookup table

Delay time ranges (in seconds)	Symbol
$[0.0, 0.2)$	C
$[0.2, 0.3)$	A
$[0.3, 0.4)$	Y
$[0.4, 0.5)$	B
$[0.5, 10.0)$	Z

After symbolization, we used the HMM inference algorithm [76] [78] to reconstruct the HMM. Our entire approach of generating data on the server to receiving data and constructing a new HMM on the client is shown in Figure 4.4. Additionally, in this figure, we show how server symbol Y is mis-symbolized as B on the client due to noise causing a sudden increase in latency.

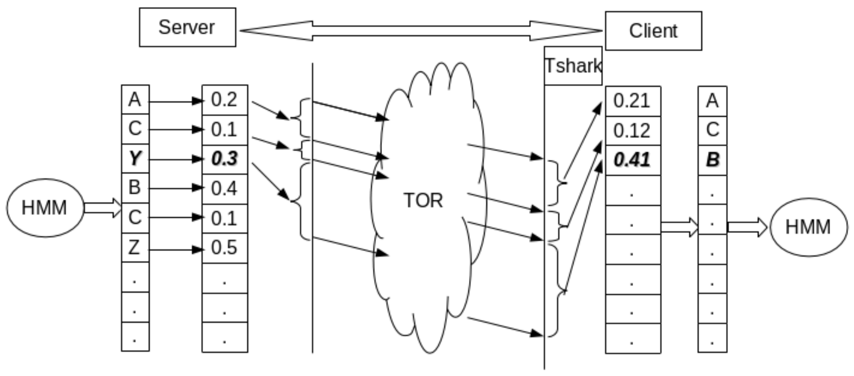


Figure 4.4: Application process.

Extra latency can cause inter-packet delays to be incorrectly symbolized. This introduces noise, which obscures the underlying protocol. Our metric, however, concentrates on the statistically significant portion of the inferred model and thus extracts the underlying protocol from the noise. In Section 4.2.2, we also show that the metric provides a quantitative measure of the noise in the Tor network.

4.2.2 Experiment Result and Analysis

We captured 100,000 packets at the client and inferred HMM M_2, shown in Figure 4.5. Due to noise in the latency values, the underlying protocol is difficult to discern. Typically, noise in the data can be mitigated by collecting more data until the underlying process overwhelms the noise. We therefore collected 200,000 packets and inferred model M_3, shown in Figure 4.6, which has clearly increased in complexity due to a larger amount of noise. We confirm this by inferring a 79 state and 274 transition model from 1 million packets. Due to the high state and transition counts, it is difficult to observe any similarities between the inferred models, and difficult to determine if the correct model is inferred.

We then applied the metric approach to the inferred models. Setting the confidence $\alpha = 0.05$, the distance between M_2 and M_3 is 3.3×10^{-4}, a very high similarity value. The equivalent portion of M_2 and M_3 is shown in Figure 4.7 as model M_4. Tracing through the paths of the model, we see that it is functionally equivalent to the original model M_1. This means the underlying protocol is actually encapsulated in the inferred models but obscured by the noise. The metric helps identify the underlying protocol in the inferred models by removing statistically insignificant noise. KLD measure does not work in this case, because both models can not accept the sequence generated by the other.

The difference between the inferred model and the original model is due to the jitter of the Tor network. The Markov metric quantifies the noise by measuring this difference into a distance value. To verify it, we added artificial noise with different probabilities in the 200,000-packet data set to infer new models and then calculated the distance between the inferred model and the original model M_1. The results are in Table 4.3. Each measured noise value is the mean of 100 runs, and is close to the actual noise value. Actual noise values fall within the 95% confidence intervals of the corresponding measured values. The results show that the metric provides an appropriate quantitative measure for the noise in the system. In this application, the distance between model M_1 and M_2 is 2.9×10^{-4} by the metric, so the noise in the 100,000-packet data is 2.9×10^{-4}, which means the noise occurs around every 3500 symbols. This measure is useful, since it can be used to determine if enough data has been collected so that the inferred model is statistically significant [75, 97]. Also, the amount of the noise relates to the detection rate of HMMs [22].

In this application, we inferred protocols tunneled through Tor network using HMM in-

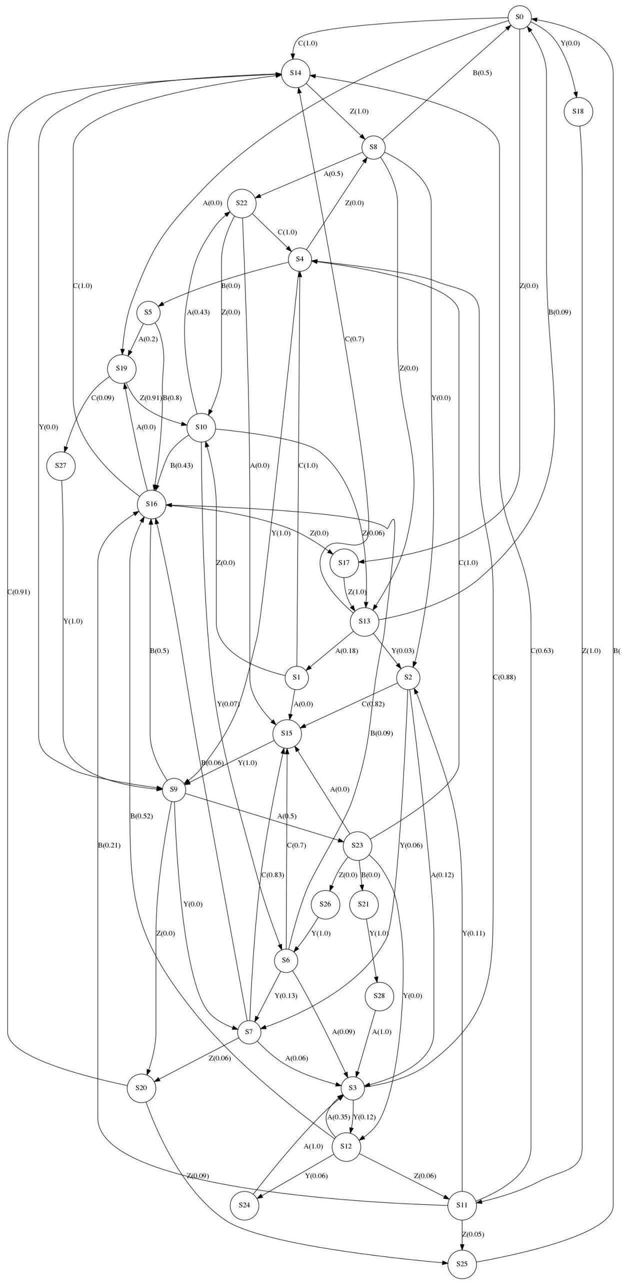


Figure 4.5: Reconstructed model M_2 from 100,000 packets.

ference approach. The Markov metric identified the correct model from inferred model which is obscured with a large amount of noise events. As a byproduct, the Markov metric also measures the noise in the Tor communications, because it quantifies the noise event into a distance value.

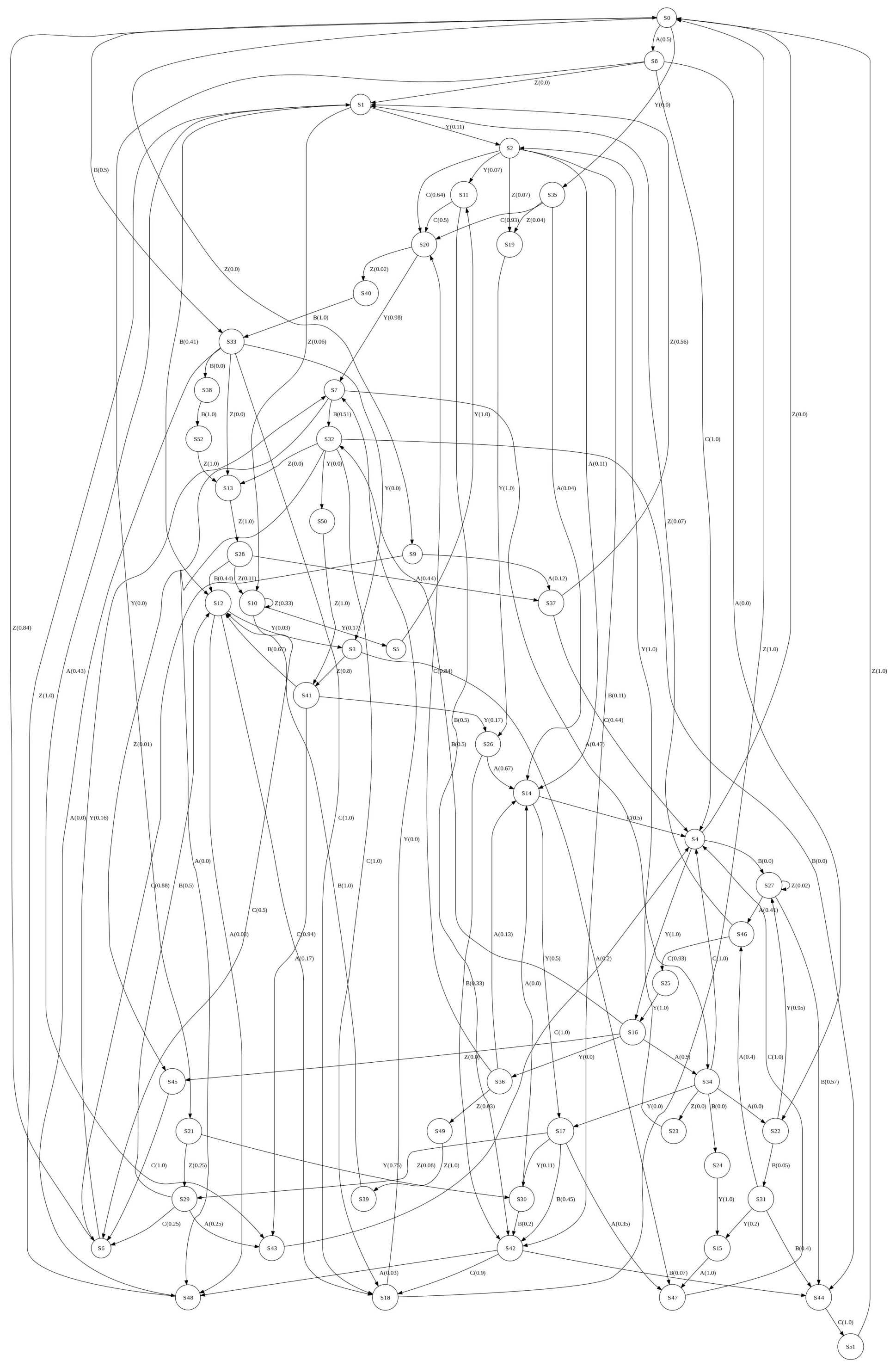


Figure 4.6: Reconstructed model M_3 from 200,000 packets.

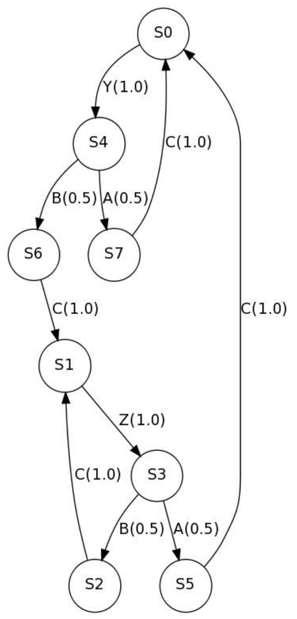


Figure 4.7: Model M_4 after pruning low probability events

Table 4.3: Noise measurement

Noise Frequency	Actual Noise Value	Measured Noise Value	95% Confidence Intervals
Every 100 symbols	1.0×10^{-2}	9.9×10^{-3}	$[0, 5.8 \times 10^{-2}]$
Every 500 symbols	2.0×10^{-3}	4.6×10^{-3}	$[0, 4.1 \times 10^{-2}]$
Every 1000 symbols	1.0×10^{-3}	2.0×10^{-3}	$[0, 2.4 \times 10^{-2}]$
Every 3000 symbols	3.3×10^{-4}	6.8×10^{-4}	$[0, 8.9 \times 10^{-3}]$

4.3 Centralized Botnet Traffic Detection

In this section, we use our HMM inference and detection approaches for centralized botnet traffic detection. Some previous research works also used traffic analysis to detect traffic patterns in botnet communications. BotSniffer [44], detects botnet traffic based on the observation that bots in a botnet will have the same traffic activity in a certain time window. Yu et al.[98] detect camouflaging worm traffic from normal traffic using a spectrum-based scheme on the traffic data. The approach in [88] examines flow characteristics such as bandwidth, packet timing, and burst duration to extract evidence of botnet C&C activity. A passive DNS traffic analysis system for detecting and tracking malicious flux networks of a botnet is proposed in [67].

In this application, we also apply traffic timing analysis, because inter-packet timings of botnets relate to botnet command and control processing time, idling time, contacting period and other botnet activity characteristics. Previous works [44, 59, 92, 20] used correlation of timing data for analysis and detection. Unlike these works, we consider the underlying communication patterns of traffic timings, since inter-packet timings relates to underlying botnet behaviors. We use hidden Markov models (HMMs) [70] to detect patterns in timing data. Furthermore, network protocols can usually be represented by finite state machines, which can be appropriately modeled by HMMs. HMMs are heavily used in pattern recognition applications, such as gesture [61], script [82], and language [69] [84] recognition, but little work has been done for botnet traffic detection.

Therefore, we use HMMs for automated network traffic analysis to detect C&C communications of centralized botnets. We focus on Zeus botnets (Zbots), one of the largest HTTP-based botnets. After collecting packet timings of Zeus botnet C&C communication traffic, we calculate timing differences between successive packets. Inter-packet delays are binned to translate them into symbolic data streams. HMMs are then inferred from these streams of symbolic timing data, using the HMM inference algorithm [76]. We also apply a model confidence test [75, 97] to check if the inferred model is representative of the underlying process.

Once HMMs are inferred, they are used to detect the botnet communications traffic. Using the HMMs confidence intervals approach [22], we detect whether or not a sequence of traffic data is botnet traffic. Experimental results on traffic data collected on real-world botnets show that this approach can dependably distinguish botnet traffic from normal traffic. We also compare our results with the autocorrelation approach in [44], which detects botnet traffic based on spatial-temporal correlation of timing data in a local network. While their work checks the periodicity of the traffic data, our approach checks underlying behaviors of the botnet. We show our approach is more accurate in detection.

4.3.1 Zeus Botnet

Zeus botnets, also known as Zbots, form a malicious network that uses key-logging and screen-capture techniques to steal sensitive data such as usernames and passwords for online banking, social networks and e-mail accounts. Zbots have existed at least since 2007, but have evolved rapidly over time. Figure 4.8 shows a diagram of reported Zbot infections from Dec. 08 to Mar. 10 [4]. Various Zeus botnets are composed of millions of compromised computers (around 3.6 million in the United States) [8]. 2,411 companies and organizations are said to have been affected by the criminal operations [65] and millions of dollars are involved in cyber banking fraud using Zbots [33]. There are malware packages readily available online for sale or to download for free. The package basically contains a builder that can generate a bot binary, a template configuration file and Web server files (PHP, SQL templates) for use as the command and control server. After the malwares are created, they are usually spread out by spamming and web phishing. In fact, there is an entire black market industry for selling, renting and managing Zbots, giving attackers great economic incentives[3].

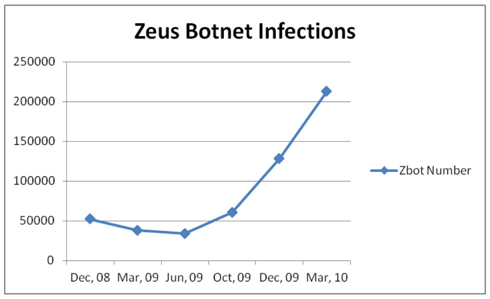


Figure 4.8: Number of reported Zbot infections

4.3.2 Zbot Traffic

Every Zbot has a centralized C&C server and the communication between the C&C server and bots uses HTTP. It is encrypted using RC4 and the key is specified by the botherder. Once a computer is infected, it sends a “GET” command to the C&C server to get the configuration file. The C&C server sends a “HTTP/200” with an “OK” code, and appends the encrypted configuration file at the end of the packet. The bot then “POST”s sensitive data to the C&C server repeatedly, following an interval specified in the configuration file. The C&C server replies each time with a “HTTP/200 OK”. New commands may also be attached to the reply. Figure 4.9 shows the communication between the bot and the C&C server. Bots of a Zbot perform similar communication and malicious activity patterns through HTTP packets.

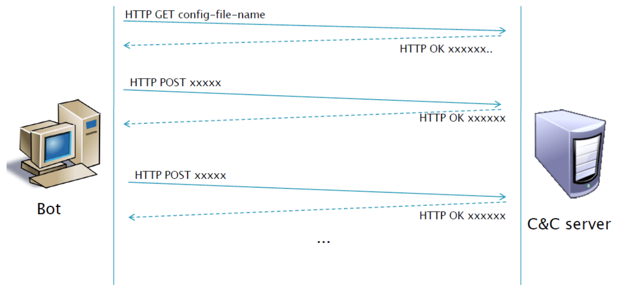
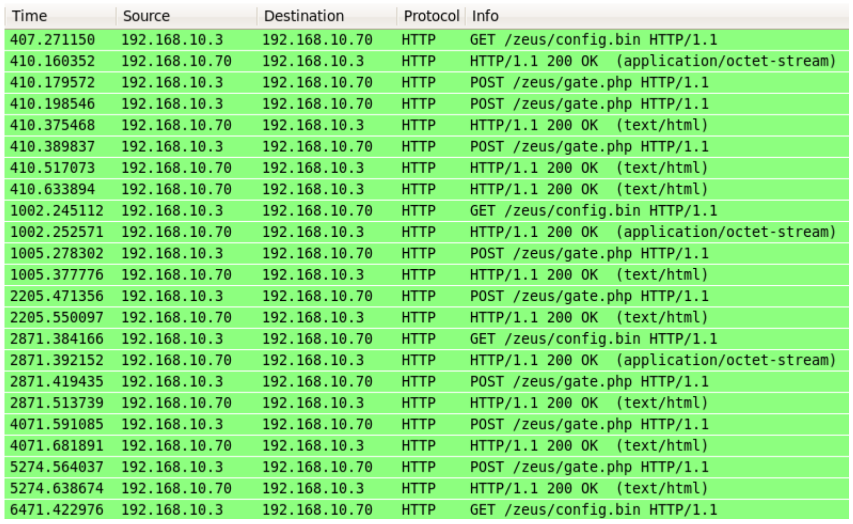


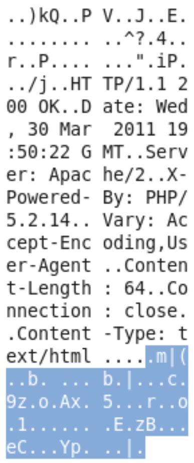
Figure 4.9: Botnet C&C communication

We set up a standalone Zeus botnet to collect traffic data. With a Zbot version 1.2.4 package, we set up a C&C server and built a bot binary with the default configuration. Using this bot binary, we infected two machines in the local network. So the local Zbot network consists one C&C server and two bots. Figure 4.10-(a) shows the captured botnet traffic in our local Zbot, and Figure 4.10-(b) shows the encrypted configuration appended at the end of a "HTTP/200 OK" reply.



```
Time         Source         Destination    Protocol Info
407.271150   192.168.10.3   192.168.10.70  HTTP     GET /zeus/config.bin HTTP/1.1
410.160352   192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (application/octet-stream)
410.179572   192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
410.198546   192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
410.375468   192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
410.389837   192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
410.517073   192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
410.633894   192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
1002.245112  192.168.10.3   192.168.10.70  HTTP     GET /zeus/config.bin HTTP/1.1
1002.252571  192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (application/octet-stream)
1005.278302  192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
1005.377776  192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
2205.471356  192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
2205.550097  192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
2871.384166  192.168.10.3   192.168.10.70  HTTP     GET /zeus/config.bin HTTP/1.1
2871.392152  192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (application/octet-stream)
2871.419435  192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
2871.513739  192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
4071.591085  192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
4071.681891  192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
5274.564037  192.168.10.3   192.168.10.70  HTTP     POST /zeus/gate.php HTTP/1.1
5274.638674  192.168.10.70  192.168.10.3   HTTP     HTTP/1.1 200 OK  (text/html)
6471.422976  192.168.10.3   192.168.10.70  HTTP     GET /zeus/config.bin HTTP/1.1
```

(a) Zbot Communication



```
..)kQ..P V..J..E.
........ ..^?.4..
r..P.... ...".iP.
../j..HT TP/1.1 2
00 OK..D ate: Wed
, 30 Mar  2011 19
:50:22 G MT..Serv
er: Apac he/2..X-
Powered- By: PHP/
5.2.14.. Vary: Ac
cept-Enc oding,Us
er-Agent ..Conten
t-Length : 64..Co
nnection : close.
.Content -Type: t
ext/html .....m|(
..b. ... b.|...c.
9z.o.Ax. 5...r..o
.1...... .E.zB...
eC...Yp. ..|.
```

(b) Encrypted configuration

Figure 4.10: Zbot Traffic

Zbot traffic data has identifiable patterns. As shown in Figure 4.10-(a), bots repeatedly communicate with the C&C server to update their configurations or get new commands. Inter-packet timings relate to command execution time, idling time, contact period and other botnet activities, therefore are a consequence of botnet behaviors. Constant communication delays in the network can be filtered out by using the time differences between successive packets. Sudden changes in communication delays may affect inter-packet timings, however, by checking model significance during HMM inference we remove the influence of this noise [75, 97]. With enough data, randomness introduced by intentional padding or the metamorphic code of the bot malware will converge to stationary probability distributions[1], which can be modeled by HMMs appropriately. Model significance also ensures enough data is used to incorporate complete behaviors of bots [75, 97]. Inter-packet timing data can be obtained easily by network sniffing without reverse engineering malware binaries or encrypted packets, and different bots of the same botnet will have similar communication patterns.

[1]Assuming the code is not updated beyond its metamorphism.

For example, bots send similar activity messages or execute the same command at a similar time intervals. Based on these advantages, we use HMMs to deduce patterns in inter-packet delays, and apply inferred models to detect similar botnet traffic.

4.3.3 Symbolization

We used Tshark [7] to capture packet timing. Since we consider the Zbot C&C channel, we only captured HTTP packets. Inter-packet delays are calculated by finding time differences between successive packets. After collecting traffic timings, we convert them into symbols. We find the local minima in the histogram of the delays to determine different ranges. Figure 4.11 shows the histogram of Zbot inter-packet delays. There are two distinct clusters so we use two symbols.

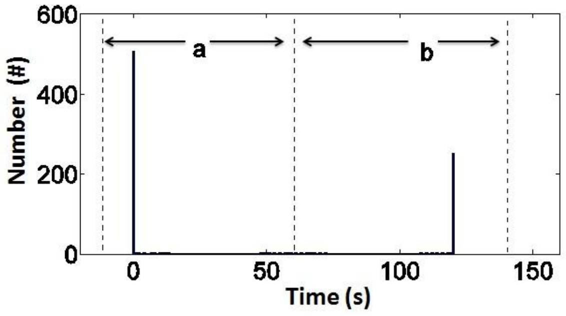


Figure 4.11: Histogram of inter-packet delays

4.3.4 Experiment

We used HMM approaches to detect Zbot traffic. The detection process is described in Figure 4.12. From Zbot traffic timing data, we infer HMMs. Using the inferred HMM, we detect whether or not a sequence of network traffic is from a Zbot based only on the timings in this traffic sequence. A white list is used to filter out trusted network flows based on IP addresses. It eliminates the number of network flows needed for detection and also reduces false alarms.

To get real-world Zbot traffic data, we downloaded Zbot binaries for different versions and C&C servers in [10] to infected virtual Windows machines which were connected to the Internet by either wireless or wired connection. We got a total of 15 different C&C botnet communications. Timing data was then captured from these communications and 15,000 packet timings were obtained. We split them into two sets: training data and testing data. From the training data, we inferred the

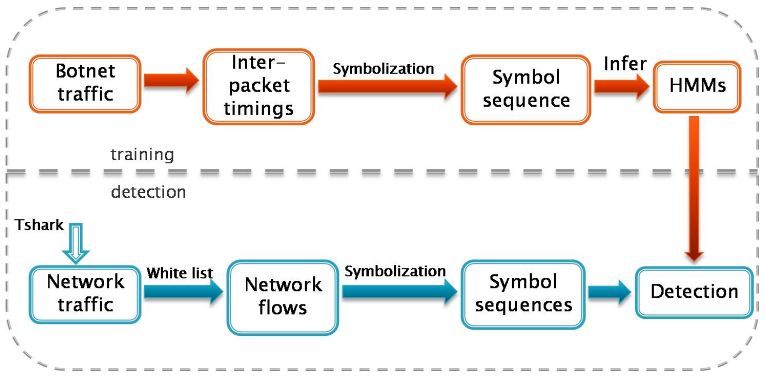


Figure 4.12: Botnet traffic detection

HMM in Figure 4.13. This HMM is statistically significant according to the model confidence test [97].

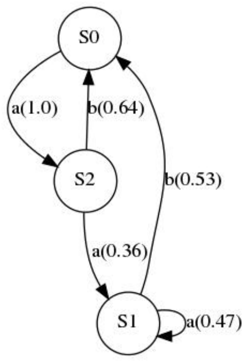
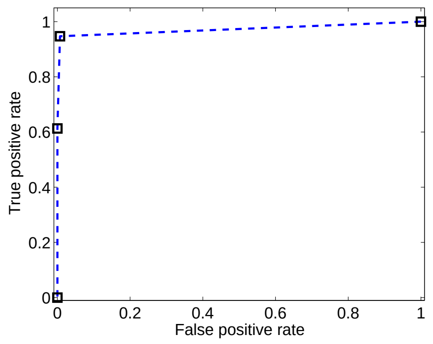


Figure 4.13: HMM of the wild Zbot

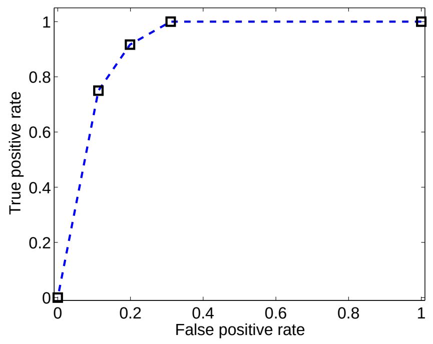
We got a portion of HTTP traffic data (only packet headers[2]) from Clemson University campus network over a time span of 40 hours. It is separated into distinct flows and timing data for each flow is symbolized respectively. Both real-world normal and testing botnet traffic data were then sent to the botnet HMM for detection. There were 150 sequences of Zbot traffic and 150 sequences of normal traffic. Each sequence contains 50 symbolized inter-packet timings of a network flow. If a Zbot traffic sequence was recognized as botnet traffic, we declared a true positive. If a normal traffic sequence was recognized as a Zbot signature, we declared a false positive. Using the confidence interval approach, we obtain the detection ROC curve in Figure 4.14-(a). Every square dot represents the result of a different threshold value.

[2]This shows another advantage of timing analysis: by just capturing headers, it eliminates much of the network adapter processing time and the detection can be performed in real-time.

We find the optimal point is $(0.947, 0.007)$ with an Euclidean distance of 0.053 to (0,1). The corresponding optimal threshold can be any value in $[0, 0.667]$, with an empirically determined 94.7% true positive rate and a 0.7% false positive rate. Thus we can dependably detect wild Zbot traffic using the proper threshold value.



(a) Zbot HMM Detection ROC curve (95% CI)



(b) Autocorrelation Detection ROC curve

Figure 4.14: Detection ROC curves

We compared our detection result with the single client detection approach from [44]. They applied autocorrelation analysis for a sequence of traffic from a single client to determine whether or not this sequence was botnet traffic. This technique encodes the traffic in a time window into a four-element vector: $< OutPkt\#, OutPktSize, InPkt\#, InPktSize >$, and uses the autocorrelation of the encoded sequence to find repeating patterns. This approach essentially checks whether or not a sequence of traffic is periodic. There are several drawbacks of this approach. First, it is hard to determine the size of the time window, since inter-packet timings may vary from milliseconds to several minutes. Inappropriate time windows may cause the encoded sequence to include incorrect patterns. Secondly, network noise can disturb the periodic packet timings and thus make them undetectable [44]. Also, some benign network services will generate periodic patterns, such as email, social network and cloud service applications, which need to update with servers periodically [44]. So merely checking the periodicity will yield more false positives and fewer true positives. This is confirmed by the detection ROC curve in Figure 4.14-(b), where the optimal detection result shown is a 91.7% true positive rate and 19.8% false positive rate on the same data set.

Because our approach checks the underlying behaviors of the botnet, not the periodicity, and the effect of network noise is removed during the process of data binning and model confidence checking. Thus, our approach has a better detection rate than the autocorrelation analysis from

[44]. For centralized botnets like Zbot, if a bot agent is found and intercepted, other bots can also be detected using our HMM detection approach and the central C&C server can be identified. Therefore, our approach provides a solution for detecting a centralized botnet.

4.4 P2P Botnet Traffic Detection

To avoid the single-point failure disadvantage of the centralized structure, hierarchical botnet uses P2P techniques [5, 9, 56, 66]. In this section, we attempt to apply stochastic grammars to P2P hierarchical botnet traffic detection. We first construct hierarchical P2P botnets using Clemson campus computer cloud (Palmetto cloud). Using the HMM detection approach, we find that HMMs can not detect recursive patterns in simulated P2P botnet traffic timings. Therefore, we applied PCFG detection approach. The experiment results show that PCFGs have accurate detection rates.

Next, we use our PCFG approach to detect real-world P2P botnet traffic. We get a trace of traffic from Storm botnet, one of the early P2P botnets [43]. We use production rules that explain the patterns in the Storm traffic. By parsing the traffic data, we estimate the production probabilities and use the estimated PCFG to detect new sequences of network traffic timings. The result shows that our approach can adequately distinguish the botnet traffic from the normal traffic.

4.4.1 Simulated Hierarchical Botnet

Hierarchical botnets are simulated using Clemson campus cloud computers. We use TCP as C&C communications to make the communication in the hierarchical botnet reliable. A C&C server is set up to control a tree-structured computer network, as shown in Figure 4.15. We use 40 nodes to construct a hierarchical botnet with 10 levels.

The C&C server sends out commands repeatedly. After receiving the command, every bot executes it, by waiting for a certain amount of time (different delays for different malicious actions are in Table 4.4), and replies back. Every node also passes the commands from its parent node to its children nodes, and passes back children nodes' responses. All forwarded messages are low-latency and different messages use separate packets. This is important, because in this way, every intermediate node can take control of sub-botnet (tree structure below it) and preserves timing patterns of all nodes in its sub-botnet, not only its direct children nodes. This also mimics the behaviors of a hierarchical botnet. Malicious actions and corresponding timing delays in Table 4.4

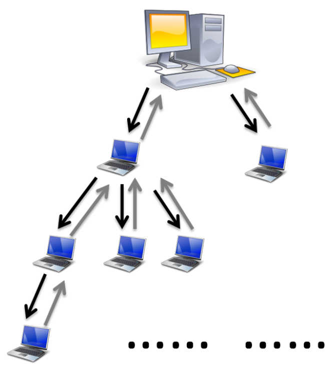


Figure 4.15: Simulated Hierarchical Botnet

are chosen for illustration. Actual botnets may have different malicious behaviors and associated timing delays.

Table 4.4: Delay time table

Malicious actions	Delay time (in seconds)
i (stealing identity)	5
s (sending spam)	10
m (installing malware)	15
d (performing DDoS attacks)	20

We simulate two hierarchical botnets based on PCFGs in Table 4.5. The first three production rules represent the hierarchical structure of a botnet. The last four productions represent the behaviors of bots in each botnet. Malicious actions follow the corresponding production probabilities. These production rules and associated probabilities are used for illustration. Actual botnets may have different structures and different probabilities. For example, P2P Zbots are mainly targeted for identify theft, therefore, the production for this malicious action of P2P Zbot will have a dominant high probability compared to other malicious actions.

After timing data is collected by Tshark [7], we use Table 4.6 to translate timing data into terminals. Different ranges are found by plotting the histogram of the data. We collected 11 data sets for each botnet and each data set contains several hundred packets.

Table 4.5: Hierarchical Botnet PCFGs

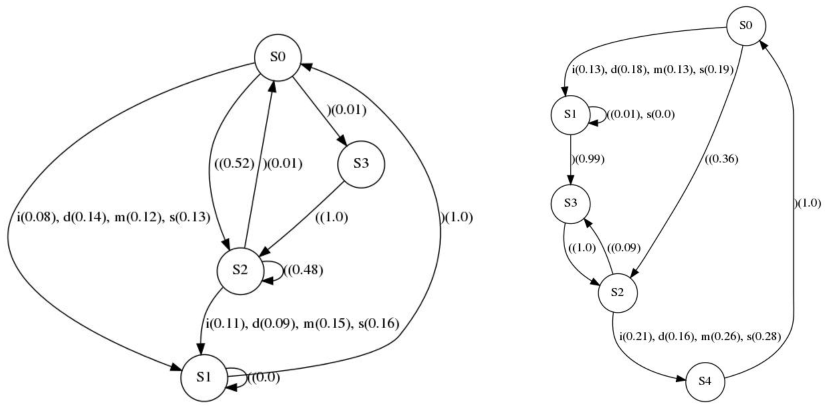
Productions	PCFG1	PCFG2
$A \rightarrow (A)$	0.3	0.4
$A \rightarrow AA$	0.3	0.2
$A \rightarrow B$	0.4	0.4
$B \rightarrow i$	0.2	0.3
$B \rightarrow s$	0.3	0.1
$B \rightarrow m$	0.3	0.2
$B \rightarrow d$	0.2	0.4

Table 4.6: Delay-Symbol lookup table

Delay time ranges (in seconds)	Symbols
$[0.0, 2.0)$	(
$[2.0, 4.0)$	)
$[4.0, 7.0)$	i
$[7.0, 13.0)$	s
$[13.0, 17.0)$	m
$[17.0, 25.0)$	d

4.4.2 Detection with HMMs

We first used 1000 symbolized inter-packet timings of simulated P2P botnet traffic to infer HMMs. The model we inferred with $L = 2$ is in Figure 4.16-(a). According to the model confidence test, this model is statistical significant. When we increase $L = 3$, we get the HMM in Figure 4.16-(b). However, the required samples for model significance is 4224, which means we need to capture more data.


S0
)(0.01)
S3
((0.52)
)(0.01)
i(0.08), d(0.14), m(0.12), s(0.13)
((1.0)
)(1.0)
S2
((0.48)
i(0.11), d(0.09), m(0.15), s(0.16)
S1
((0.0)
S0
i(0.13), d(0.18), m(0.13), s(0.19)
S1
((0.01), s(0.0)
)(0.99)
((0.36)
S3
)(1.0)
((1.0)
((0.09)
S2
i(0.21), d(0.16), m(0.26), s(0.28)
S4

(a) $L = 2$ (b) $L = 3$

Figure 4.16: Models inferred with 3000 packets

Usually, the model significance can be achieved by using more data. However, when we repeat the process with 2000 packets, the required number of samples for the newly inferred model increases to 8268 samples. Oddly, the required number of samples keeps increasing as we increase the number of samples to infer the HMM, as shown in Figure 4.17. The steady increase suggests we are not able to capture enough data to rebuild the significant model. Using the metric, we find the distances between each pair of HMMs are 1, which means the differences between them are not because of network noise and they are totally different models.

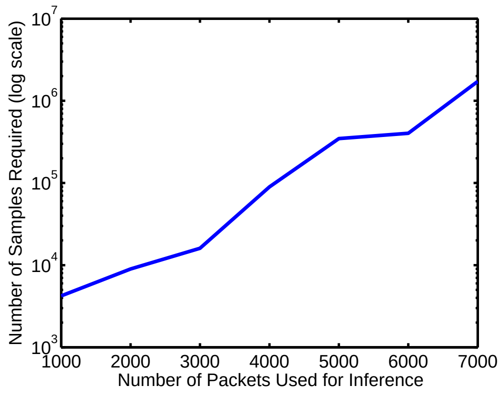


Figure 4.17: Samples required (in log scale)

Therefore, we are forced to stop HMM inference and use the approximate model inferred with $L = 2$. We then applied the confidence interval approach to detect 100 sequences of hierarchical botnet traffic timings and 100 sequences of normal traffic timings. Each sequence contains 50 symbolized inter-packet timings. If a hierarchical botnet traffic sequence is recognized as botnet traffic, we have a true positive. If a traffic sequence that is not generated by the hierarchical botnet, but is recognized as is, we have a false positive. By varying the threshold values, we obtain the detection ROC curve in Figure 4.18.

The ROC curve is around the diagonal line (the red dotted line in Figure 4.18), which means that if we want a high true positive rate, the false positive rate is also high; if we want a low false positive rate, the true positive rate is low as well. Therefore we are unable to get a good detection rate (i.e. high true positive rate and low false positive rate) using the HMM approach.

This result shows that when the data contains hierarchical and recursive patterns, HMMs are not appropriate models to use, because HMMs are stochastic regular grammars [13, 64], which has finite state limitation and can only represent patterns in regular expressions.

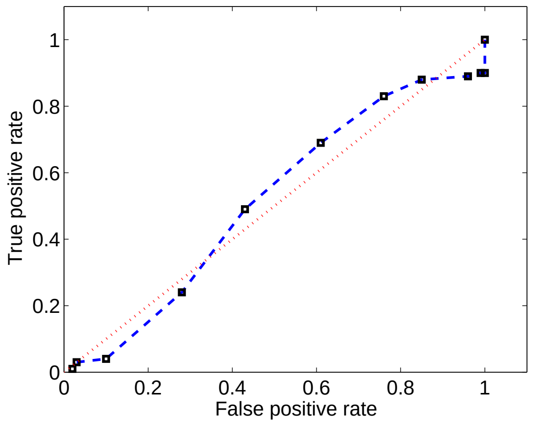


Figure 4.18: Botnet ROC curve (95% CI)

4.4.3 Detection with PCFGs

We build the LALR parser from production rules in Table 4.5, and parse the symbolized sentences to estimate production probabilities for each data set. Using the χ^2 test, we detected whether two data sets are generated from the same source PCFG (meaning that they are generated from the same hierarchical botnet). If two sets are generated by different PCFGs, but they are detected as equivalent, we have a false positive. If two data sets are generated by a same PCFG, and the χ^2 test accepts that they are equivalent, we count it as a true positive. By testing each pair of these 22 data sets, we get the detection result in Table 4.7. With a high true positive rate and a low false positive rate, we can dependably detect hierarchical botnet traffic using our detection approach.

Table 4.7: Botnet Detection Result

Data Set Source	PCFG1	PCFG2
TP rate	96.7%	93.4%
FP rate	1.6%	1.6%

For hierarchical botnet traffic, every intermediate node includes timing patterns from its sub-botnet. Therefore, its communications have recursive patterns similar to tree structures. Furthermore, traffic timing patterns of different bots in the same hierarchical botnet are also different, since traffic from different bots go through different numbers of levels, and thus have different traffic timing patterns. Therefore, using HMMs to model hierarchical traffic patterns is not appropriate.

PCFGs can expressively recognize recursive timing patterns, by parsing the symbolized timing data. The stochastic behaviors of the hierarchical botnet (i.e. infection rate, possible malicious actions, etc.) can also be modeled by production probabilities. Compared to HMMs, PCFGs are more appropriate to represent traffic patterns of hierarchical P2P botnet traffic.

4.4.4 Storm Botnet

The previous section illustrated the performance of the PCFG detection approach on the traffic data of simulated hierarchical P2P botnets. In this section, we apply PCFG to detect real-world P2P botnet traffic. We got a trace of traffic from Storm botnet [43], one of the known P2P botnets. With our PCFG detection approach, we can appropriately differentiate Storm botnet traffic from normal P2P traffic.

Storm botnet is one of the earliest botnets to use P2P techniques [43]. Storm is capable of a variety of malicious activities, such as spamming, DDoS attacks and identity theft [29]. It was estimated to include millions of infected computers in 2007 [49]. At one point, it accounted for 8% of all malware on Windows systems [38].

The communication between bots in the Storm botnet is based on the Overnet protocol, a P2P protocol that allows file sharing. The transient, "self-healing" features of the P2P network makes the botnet more resilient to monitoring and takendown actions. After infection, a bot first contact a list of static peers in the configuration file to join the network. After successful participation, it gets updated files and tools from other bots and maintains a list of neighbour peers that it repeatedly publicize with [87]. To secure the communication between bots, it encrypts the traffic [87].

In a Storm botnet, different bots may have quite different communication patterns, because some bots may have few neighbour bots while others may have a lot of neighbour bots. However, their communications may have similar or equivalent probability distributions for malicious actions. For example, if the botnet is performing DDoS attacks, a large portion of the bot's traffic will be fast and repeated packets, and this portion among bots are usually similar no matter how many neighbours a bot may have. Based on these observations, we develop productions that explains Storm botnet traffic patterns. By parsing the training data (the collected traffic that are known from a Storm botnet), we map the data into production probabilities. With the estimated PCFG, we are able to detect whether or not a new observed traffic data set is from the Storm botnet based on our detection approach.

4.4.5 Traffic Timing Analysis

We also use inter-packet timing delays as traffic patterns for Storm botnet. For P2P botnets, inter-packet timings relate to malicious action timings, publicizing periods and other botnet activity characteristics. For example, while sending spam or performing DDoS attacks, the inter-packet timings are small since bots are sending packets very fast. During normal time, the inter-packet timings are large since the bot will publicize itself with a certain time period. Therefore, inter-packet timings reflect the behaviors of a botnet.

In our application, since Storm traffic uses P2P, we are only interested in timings of UDP packets. Similarly, we translate timing data into symbols (terminals), since the PCFG detection works with sentences of terminals. Different groups of delays can be found by plotting the histogram of the data and similar delays are grouped together and replaced with a terminal.

In this application, we get a real-world Storm traffic trace from [43], where there are 13 bots in the set up. We extract P2P communication traffic for each bot and obtained 13 timing data sets, each containing several thousand timing delays. Figure 4.19 shows the histogram of inter-packet delays of a data set. There are three distinct clusters (short, medium, large delays), so we use three terminals. Table 4.8 shows different ranges for timing delays and corresponding terminals.

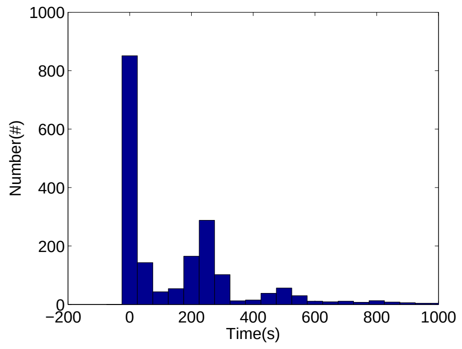


Figure 4.19: Storm botnet traffic timing histogram

Table 4.8: Delay-Terminal lookup table

Delay time ranges (in seconds)	Terminals
$[0, 150)$	a
$[150, 350)$	b
$[350, \infty)$	c

4.4.6 Storm Botnet Traffic Detection

For Storm traffic, we use the productions in Table 4.9 to explain Storm botnet traffic patterns. The first set of productions are used to represent recursive patterns in the traffic. The second set of productions represent some typical communication patterns, such as continuous short (or medium) delays, a short delay followed by a medium (or large) delay and etc. The last set of productions are used to corporate all other patterns, in case some noise is introduced in the data. With this CFG, we can build a LALR parser to parse the symbolized traffic data sets.

Table 4.9: CFG for Storm botnet traffic

Productions		
$A \rightarrow AA$	$B \rightarrow aa$	$C \rightarrow a$
$A \rightarrow B$	$B \rightarrow ab$	$C \rightarrow b$
$A \rightarrow C$	$B \rightarrow ac$	$C \rightarrow c$
	$B \rightarrow bb$	
	$B \rightarrow cc$	

We use three of the Storm traffic data sets as the training data. The rest 10 data sets are then used as test sets. After symbolization, we parse the training data sets to estimate a PCFG. This maps the probability distribution from the data set to productions. The estimated PCFG is in Table 4.10.

Table 4.10: Estimated PCFG from Storm botnet traffic

Productions					
$A \rightarrow AA$	0.500	$B \rightarrow aa$	0.499	$C \rightarrow a$	0.001
$A \rightarrow B$	0.357	$B \rightarrow ab$	0.259	$C \rightarrow b$	0.682
$A \rightarrow C$	0.143	$B \rightarrow ac$	0.073	$C \rightarrow c$	0.317
		$B \rightarrow bb$	0.095		
		$B \rightarrow cc$	0.072		

To test the performance of our detection approach, a trace of real-world background normal traffic are captured from Clemson University campus network with Tshark [7]. 10 nodes with most UDP traffic are chosen and their timing data is extracted. So we got 10 timing data sets of normal traffic, each containing several thousand packets.

Using the χ^2 test, we detect whether a data set matches the estimated PCFG. If a data set matches the PCFG, we conclude that this data set generated by the Storm botnet. For 10 test sets of the Storm botnet traffic, the χ^2 test accepted 5 data sets. The χ^2 statistics of the 5 Storm data sets that were rejected are 24.0, 18.3, 19.2, 20.4, 28.3, which are close to the critical value

$\chi^2_{0.05,df=8} = 15.5$. If we further investigate the estimated production probabilities from these data sets, we find that they are actually very close to the PCFG in Table 4.10. One example is shown in Table 4.11. Each probability differs by at most 0.03.

Table 4.11: Estimated probabilities of one data set that is not accepted

Productions					
$A \to AA$	0.500	$B \to aa$	0.495	$C \to a$	0.004
$A \to B$	0.377	$B \to ab$	0.246	$C \to b$	0.683
$A \to C$	0.123	$B \to ac$	0.070	$C \to c$	0.313
		$B \to bb$	0.080		
		$B \to cc$	0.101		

However, for 10 normal traffic data sets, none are accepted as equivalent with the estimated PCFG. The χ^2 statistics of normal traffic data sets are much larger than those of Storm traffic data sets, with a minimum value of 254.4 and a maximum value of 21203.8. Using the χ^2 critical value as a threshold and varying it, we plot the ROC curve for detection. The curve is shown in Figure 4.20. It has optimal detection rate 100% TP rate and 0% FP rate, using any threshold value between 28.3 and 254.4. Therefore, by using ROC curves, we can dependably distinguish Storm botnet traffic from normal background traffic.

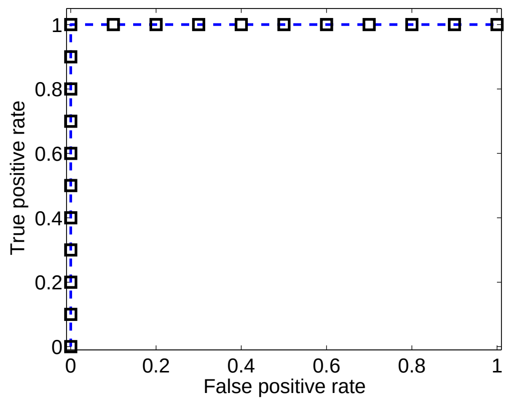


Figure 4.20: Storm botnet traffic detection ROC curve

Generally, with larger sample sizes, the χ^2 test will give more accurate detection rates. However, in this case, the Storm traffic inflates the χ^2 test values when sample sizes are very large, and thus erroneously reject data samples [74]. Network traffic are non-Gaussian, heavy-tailed processes [21, 73, 45], which are not independent and identically distributed, therefore, the estimated probability distributions are likely to not satisfy the assumptions of the χ^2 test. Since the number of training data sets is large, it is also possible that the approach is over trained and does not generalize appropriately. Another possible reason for the χ^2 test rejecting 5 Storm traffic data sets is because the production rules in Table 4.9 are estimated by the experimenter. The correct production rules may be different. Using productions inferred directly from the data might be more appropriate. However, the χ^2 test results we present are still promising. Improving the TP rate of the χ^2 test and production inference from traffic data are left as future work.

4.5 Summary

In this chapter, we applied our stochastic grammar detection approaches to network traffic analysis. We performed three application, Tor network traffic analysis, centralized botnet traffic detection and P2P botnet traffic detection.

To analyze the traffic tunneled through Tor, we set up a private Tor network and a server talks to a client through Tor based on a protocol represented by a HMM. From the timing data collected at the client, we used HMM inference algorithm to infer the protocol. However, the noise in the network obfuscates our inference process. We then apply the metric to the inferred models and find the underlying protocol is actually encapsulated in the inferred models. Also, the Markov metric quantifies the noise in the Tor communication. Our Markov metric is better than the KLD measure, because our metric is a true metric and can always return a proper distance value.

We applied HMM approach to centralized botnet traffic detection. We used Zeus botnet as an example. From inter-packet timings of Zbot C&C communications, we inferred HMMs and used inferred HMMs to detect new botnet communication traffic. Experimental results show that we can adequately detect botnet traffic from normal traffic. Our approach also outperforms the autocorrelation approaches usually performed on the network timing data, because HMMs checks the underlying system behaviors, not just the periodicity.

To detect more advanced hierarchical botnets, which use P2P techniques, we used PCFGs for detection. Using traffic data from a simulated hierarchical botnet, we showed that HMMs are not suited for detecting recursive patterns. With PCFG detection, we detected real-world network traffic timing data from Storm botnet. The χ^2 test results are promising, and performs better than HMMs. But more future work is required. Using ROC curve analysis, we can accurately detect Storm P2P botnet traffic.

Chapter 5

Conclusions

In this chapter, we give a summary of our work and achievements. We also suggest some possible future research utilizing the approaches proposed in this thesis.

5.1 Conclusion

In this thesis, we proposed pattern detection approaches with stochastic grammars for traffic timing analysis. We developed inference and detection algorithms for hidden Markov models and probabilistic context-free grammars. With these approaches, we inferred the protocols tunneled through Tor, measured the noise in the Tor communication and detected traffic that are generated from centralized and P2P botnets.

From a sequence of observations, we inferred the HMMs without any *a priori* information about the structure and initial transition probabilities. A model confidence test was used to check the significance of the inferred HMM. We extended the confidence interval approach for HMMs to detect whether or not a new sequence of observations matches with a given HMM.

To compare the similarity of HMMs, we propose a normalized statistical metric space. Our Markov metric compares HMM based on underlying system statistics, which is fundamentally different from the graph isomorphism problem from graph theory. Using the χ^2 test, we determine the equivalence of two HMMs. If they are not equivalent, the Markov metric removes states and transitions until the models are equivalent within a given statistical significance. This measures the distance between two models. Compared to KLD, which is a widely used HMM similarity measurement, our approach is more practical and provides a true metric space.

HMMs are probabilistic state machines, which are the simplest models of computation. Therefore we extended the detection approach to use PCFGs, which have more expressive power in pattern representations. We proposed a simple statistical approach for PCFG detection. We estimated production probabilities from either tree sets, or sentence sets with LALR parser. The χ^2 test was used to determine whether a data set matches a given PCFG, or whether two data sets are equivalent. From illustrative examples, our approach has better detection rates compared with the previously used inside-outside algorithm.

Three applications demonstrated the performance of our detection approaches. In the Tor traffic analysis, we used HMM inference approach to infer HMMs from a real-world application, and used Markov metric to analyze inferred models. With the metric, we were able to identify a protocol tunneled through Tor and distinguish it from statistically insignificant noise. Also with the metric, we were able to measure the amount of noise in the Tor network.

Using HMM inference and detection approaches, we presented a centralized botnet traffic detection application. We inferred HMMs from Zbot traffic timing data. The inferred HMM detects botnet traffic using confidence intervals of the state probabilities. Experimental results on real-world network traffic show that this approach appropriately differentiates botnet traffic from normal traffic, and has a higher true positive rate and a lower false positive rate than the autocorrelation analysis.

Hierarchical P2P botnets have arisen to avoid the disadvantages of centralized botnets. HMMs failed to detect recursive and structural patterns in P2P botnet traffic. We used PCFGs to detect P2P botnet traffic. From traffic timing data of a real-world Storm botnet, we estimated PCFGs, which maps probability distributions from traffic data into production probabilities. Detection results show that based on ROC curve analysis on χ^2 statistics, we can appropriately differentiate Storm botnet traffic from normal P2P traffic.

5.2 Future Research

In general, our detection approach works with timing profiles, which exist in all automated network processes. Remove timing profiles is difficult and not practical. Therefore future works can extend our detection approaches to detect other network processes, such as other malicious network processes, or certain specific network protocols.

The Markov metric is useful in behavior recognition and detection. Models that have similar statistics can be recognized and grouped together to get a larger training sample. An interesting future application based on this could be analyzing consumer behaviors. In [22], Brooks et.al constructed HMMs for purchase behaviors of different consumers from Netflix challenge data set [16]. Using the metric we proposed, it is possible to identify consumers who have similar purchase behaviors and improve the product recommendation systems.

For probabilistic context-free grammars, future research could learn production rules from observations. Inferred productions may be more appropriate for representing underlying processes. The detection rates may improve.

We could also create a metric space for PCFGs. Although it is undecidable if two context-free grammars generate the same language [46], graph matching is also complex, future research could adapt our Markov metric to determine if the statistics of two PCFGs are equivalent. This could be useful for behavior recognition and pattern detection.

Appendices

Appendix A Curriculum Vitae

Education

- **Ph.D**, Electrical Engineering, Clemson University, Aug 2009-Dec 2012
  - **Advisor:** Dr. Richard Brooks
  - **Dissertation topic:** Network Traffic Analysis using Stochastic Grammars
  - **GPA:** 4.0/4.0
  - **Minor:** Math
- **Bachelor of Science**, Shanghai Jiao Tong University, China.P.R, Jun 2009
  Major: Automation, **Minor:** Law

Experience

- Research Assistant, ECE, Clemson University Aug 2009-Dec 2012
  - Pattern detection using stochastic grammars (Hidden Markov Models, Probabilistic Context-Free Grammars) for network security
  - Network traffic data analysis for SSH, Tor and Botnet
- Teaching Assistant, ECE449/649 Computer Network Security, Fall 2011
  - Administrate Linux systems and wired/wireless network;
  - Demonstrate buffer overflow exploits and virus implementation.
- Summer Research Assistant, ME, Clemson University May 2011-Aug 2011
  - Design user interface for finite element analysis tool;
  - Integrate C++, Java and Java Web Start projects.
- Research student, Shanghai Jiao Tong University Spring 2009
  - Use Active Appearance Models for face recognition;
  - Optimize the algorithms and improving the precision of landmark location.

Publications

- **Chen Lu**, J. Schwier, R. Craven, L. Yu, R. Brooks, and C. Griffin, "A Normalized Statistical Metric Space for Hidden Markov Models", *IEEE Transactions on System Man and Cybernetics, Part B: Cybernetics,* 2012. **(In press)**
- **Chen Lu**, R. Brooks, and C. Griffin, "Botnet Traffic Detection Using Hidden Markov Models", *IEEE Transactions on Dependable and Secure Computing,* **Under review (2012)**.
- **Chen Lu**, R. Brooks, "Pattern Detection Using Probabilistic Context-Free Grammars", **Under preparation for submission**.
- **Chen Lu**, J. Schwier, R. Brooks, C. Griffin, S. Bukkapatnam, "Markov Model Inferencing in Distributed Systems", *Chapter of "Distributed Sensor Networks" Second Edition*, S. S. Iyenar, and R. R. Brooks, ed.'s, Chapman & Hall, Boca Raton, FLA, 2012.
- **Chen Lu**, R. Brooks, "Timing Analysis in P2P Botnet Traffic Using Probabilistic Context-Free Grammars", *8th Annual Workshop on Cyber Security and Information Intelligence Research*, ACM, Oct. 2012.
- **Chen Lu**, R. Brooks, "P2P Hierarchical Botnet Traffic Detection Using Hidden Markov Models", *Learning from Authoritative Security Experiment Results (LASER) Workshop,* Jul. 2012.
- **Chen Lu** and R. Brooks, "Botnet Traffic Detection", *Kasperskys First "IT Security for the Next Generation" American Cup Paper*, Nov. 2011.
- **Chen Lu** and R. Brooks, "Botnet Traffic Detection Using Hidden Markov Models", *7th Annual Workshop on Cyber Security and Information Intelligence Research*, ACM, Oct. 2011.
- **Chen Lu** and R. Brooks, "Applications of Side Channel Attacks", *ARCTIC Information Assurance Workshop Proceedings*, Washington DC, Aug. 2011.

Awards

- 2011, Kaspersky First "IT Security for the Next Generation" American Cup Finalist
- 2011, US DOT Connected Vehicle Challenge Winner (as a team member)
- 2009, Third place in Cypress PSoC China Innovator Design Challenge

Proficiency

- Java, C/C++, Python, MATLAB, LaTeX, R
- Linux/Mac/Microsoft Windows Family, Wireshark, VMWare

Leadership

- Research group/lab administrator, Clemson University. 2009-2012
    - Administrate distributed Linux systems;
    - Arrange and manage equipments;
    - Video Design for research/lab projects.
- Webmaster for Chinese Student Association (CSSA), Clemson University, 2010-2012
    - Administrate CSSA official website;
    - Design and develop the public bulletin board system (BBS) for CSSA.

Appendix B The Construction of LALR Parse Table

We introduce the process of generating a LALR parse table from [13]. We first need to augment a CFG G to G'. If G is a CFG with start symbol S, then the augmented grammar G' is G with a new start symbol S' and an additional production $S' \to S$. The augmented grammar G' is then used for this LALR parser construction method. Algorithm B.1 describes the construction of a LALR table.

Algorithm Description B.1 – Construction of LALR table

Input: An augmented CFG G'.
Output: A LALR parsing table with functions ACTION and GOTO for CFG G';

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(1) items, using the approach in B.2;
2. Replace all sets having the same core items by their union;
3. Let $C' = \{J_0, J_1, \ldots, J_m\}$ be the new collection. State i of the parser is generated from J_i:
   (a) If $[A \to \alpha \cdot a\beta, b]$ is in J_i and GOTO$(J_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j";
   (b) If $[A \to \alpha\cdot, a]$ is in J_i, $A \neq S'$ then set ACTION$[i, a]$ to "reduce $A \to \alpha$";
   (c) If $[S' \to S\cdot, \$]$ is in J_i, where \$ is the endmark, then set ACTION$[i, \$]$ to "accept";
4. If J_i is in the union of one or more LR(1) sets, say $J_i = I_1 \cup I_2 \cup \ldots$, then the core of GOTO$(I_1, A)$, GOTO$(I_2, A)$, ...are the same. Let J_j the union of all sets of items having the same core as GOTO(I_1, A), then GOTO$(J_i, A) = J_j$.

We discuss how to generate LR(1) items for CFGs using the way in [13]. An LR parser is also in a finite state format. States represent sets of "items." An item of a grammar is a production with a dot in the right hand side. For example, the production $A \to XYZ$ yields the following items:

$$\begin{aligned} &A \to \cdot XYZ \\ &A \to X \cdot YZ \\ &A \to XY \cdot Z \\ &A \to XYZ\cdot \end{aligned} \tag{1}$$

where A is a nonterminal and X, Y, Z are grammar symbols. An item indicates how much of a production has been parsed and what is expected.

To get LR(1) items, we use the FIRST and FOLLOW functions. The FIRST(α) function, where α is string of grammar symbols, generates a set of nonterminals that begin strings derived from α. So if a terminal a is in FIRST(α) set, there exists a derivation that $\alpha \overset{*}{\Rightarrow} a\beta$. The FOLLOW$(A)$ function, where A is a nonterminal, generates a set of nonterminals that appears immediately to the right of A in some sentential form. So if a terminal a is in FOLLOW(A) set, there exists a derivation that $S \overset{*}{\Rightarrow} \alpha A a \beta$, for some α and β. Detailed algorithms to get FIRST(α) and FOLLOW(A) sets are discussed in [13] .

An LR(1) item is $[A \to \alpha \cdot \beta, a]$, where $A \to \alpha\beta$ is a production and a is a terminal in FOLLOW(A). The 1 in LR(1) refers to the length of the second component, the *lookahead* of the item. To construct collection LR(1) items, we use an augmented grammar. The following algorithm is used to construct a collection of sets of LR(1) items.

Algorithm Description B.2 – Construction of LR(1) item sets C

```
CLOSURE:
  • For each item $[A \to \alpha \cdot B\beta, a]$ in $I$:
      – For each production $B \to \gamma$ in CFG $G$:
          ∗ For each terminal $b$ in FIRST($\beta a$):
              · Add $[B \to \cdot\gamma, b]$ to set $I$;
  • Return $I$ when no more items are added to $I$.
GOTO($I, X$):
  • Initialize set $J$ to be empty:
  • For each item $[A \to \alpha \cdot X\beta, a]$ in $I$:
      – Add item $[A \to \alpha X \cdot \beta, a]$ in $I$;
  • Return CLOSURE($J$).
Item sets $C$:
  • Initialize $C$ to CLOSURE($\{[S' \to \cdot S, \$]\}$), where \$ is the endmark;
      – For each set of items $I$ in C:
          ∗ For each grammar symbol $X$:
              · If GOTO($I, X$) is not empty and not in $C$:
              · Add GOTO($I, X$) to $C$;
  • Return $C$ when no new sets of items are added to $C$.
```

Appendix C HMM Inference Algorithms

Given a sequence $\mathbf{y}$ of output observations, the Causal State Splitting and Reconstruction (CSSR) Algorithm [78, 77, 80] infers a set of causal states and a transition structure for a HMM that provides a minimum entropy estimation of the true underlying process dynamics. This algorithm requires an parameter L, which is large enough to indicate the current state based on the history sequence with length L. The CSSR Algorithm is described as below:

Algorithm Description C.1 – CSSR Algorithm from [77] and [76]

```
Input: Observed sequence $\mathbf{y}$; Alphabet $\mathcal{A}$, Integer $L$;
Initialization:
  1. Define state $q_0$ and add $\lambda$ (the empty string) to state $q_0$. Set $Q = \{q_0\}$.
  2. Set $N := 1$.
Splitting (Repeat for each $i \leq L$)
  1. Set $W = \{\mathbf{x} | \exists q \in Q(\mathbf{x} \in q \wedge |\mathbf{x}| = i-1)\}$. The set of strings in states of the current model with length
     equal to $i-1$.
  2. Let $N$ be the number of states.
  3. For each $\mathbf{x} \in W$, for each $a \in \mathcal{A}$, if $a\mathbf{x}$ is a subsequence of $\mathbf{y}$, then
     (a) Estimate $f_{a\mathbf{x}|\mathbf{y}} : \mathcal{A} \to [0,1]$, the probability distribution over the next input symbol.
     (b) Let $f_{q_j|\mathbf{y}} : \mathcal{A} \to [0,1]$ be the joint state conditional probability distributions; that is, the prob-
         ability given the system is in state $q_i$, that the next symbol observed will be $a$. For each $j$,
         compare $f_{q_j|\mathbf{y}}$ with $f_{a\mathbf{x}|\mathbf{y}}$ using an appropriate statistical test with confidence level $\alpha$. Add $a\mathbf{x}$
         to the state that has the most similar probability distribution as measured by the $p$-value of the
         test. If all tests reject the null hypothesis that $f_{q_j|\mathbf{y}}$ and $f_{a\mathbf{x}|\mathbf{y}}$ are the same, then create a new
         state $q_{N+1}$ and add $a\mathbf{x}$ to it. Set $N := N+1$.
Reconstruction
  1. Let $N_0 = 0$.
  2. Let $N$ be the number of states.
  3. Repeat while $N_0 \neq N$:
     (a) For each $i \in 1, \dots, N$: Set $k := 0$. Let $M$ be the number of sequences in state $q_i$. Choose a
         sequence $\mathbf{x}_0$ from state $q_i$. Create state $p_{ik}$ and add $\mathbf{x}_0$ to it. For all sequences $\mathbf{x}_j$ ($j > 0$) in
         state $q_i$:
         i. For each $a \in \mathcal{A}$ $\mathbf{x}_j a$ produces a sequence that is resident in another state $q_k$. Let $(\mathbf{x}_j, a, q_k) \in$
            $\delta$.
         ii. For $l = 0, \dots, k$, choose $\mathbf{x}$ from sequences within $p_{ik}$. If $\delta(\mathbf{x}_j, a) = \delta(\mathbf{x}, a)$ for all $a \in \mathcal{A}$, then
             add $\mathbf{x}_j$ to $p_{ik}$. Otherwise, create a new state $p_{ik+1}$ and add $\mathbf{x}_j$ to it. Set $k := k+1$.
     (b) Reset $Q = \{p_{ik}\}$; recompute the state conditional probabilities $f_{q|\mathbf{y}}$ for $q \in Q$ and assign transi-
         tions using the $\delta$ functions defined above.
     (c) Let $N_0 = N$.
     (d) Let $N$ be the number of states.
  4. The model of the system has state set $Q$ and transition probability function computed from the $\delta$
     relations and state conditional probabilities.
```

The zero knowledge HMM identification algorithm [76] extends the CSSR algorithm to find the parameter L automatically. This algorithm is described as below:

Algorithm Description C.2 – Zero Knowledge HMM Identification Algorithm from [76]

Input: Observed sequence $\mathbf{y}$; Alphabet $\mathcal{A}$;

Initialization:

1. Set $L = 1$.
2. The set $Q_{L-1} = \{q_0\}$ and $G_{L-1} = \langle Q_{L-1}, \mathcal{A}, \delta_{L-1}, p_{L-1}\rangle$, where $(q_0, y, q_0) \in \delta_{L-1}$ for all $y \in \mathcal{A}$ and $p_{L-1}(q_0, y, q_0)$ is the proportion of times symbol y occurs in history $\mathbf{y}$. (This is the ϵ-machine that results when CSSR Algorithm is run with $L = 0$.)
3. Let the length of $N = |\mathbf{y}|$.

Main Loop:

1. Let $G_L = \langle Q_L, \mathcal{A}, \delta_L, p_L\rangle$ be the ϵ-machine output of CSSR Algorithm with $\mathbf{y}$, $\mathcal{A}$ and L;
2. For every state $q_0 \in Q_L$, record the sequence of states $\mathbf{q}_L^{q_0} = \{q_1, q_2, \ldots, q_N\}$ that occurs when $\hat{\delta}_L$ is recursively applied with input $\mathbf{y}$ starting at state q. That is, $q_1 = \hat{\delta}_L(q_0, y_1)$, $q_2 = \hat{\delta}_L(q_1, y_2)$ etc. If there is some $i \leq N$ for which $q_i =\uparrow$, then we discard sequence $\mathbf{q}_L^{q_0}$ as undefined.
3. Each sequence $\mathbf{q}_L^{q_0}$ defines a partial function $f_L^{q_0} : [N] \times \mathcal{A} \to Q_L$. If q_k is the k^{th} element of sequence $\mathbf{q}_L^{q_0}$, then $f_L^{q_0}(k, y_k) = q_k$. That is, position k with symbol y_k is associated to state q_k. Let $\mathcal{F}_L$ be the set of functions $f_L^{q_0}$ defined in this way.
4. Compare the functions in $\mathcal{F}_L$ to the elements of $\mathcal{F}_{L-1}$: We will use these sets to define a matching problem whose optimal solution will be used to define a stopping criterion.
    (a) Let $\mathcal{I}$ be a set of indices corresponding to elements of Q_{L-1} and $\mathcal{J}$ be a set of indices corresponding to elements of Q_L.
    (b) Define binary variables x_{ij} ($i \in \mathcal{I}$, $j \in \mathcal{J}$). We will declare $x_{ij} = 1$ if and only if state q_i of Q_{L-1} is matched with state q_j of Q_L.
    (c) Define the following coefficients:
    $$r_{ij} = \sum_{q_{0_L} \in Q_L, q_{0_{L-1}} \in Q_{L-1}} \left| (f_L^{q_{0_L}})^{-1}(q_i) \cap (f_{L-1}^{q_{0_{L-1}}})^{-1}(q_j) \right|.$$
    (d) Solve the Matching Problem:
    $$\max_{x_{ij}} \; m_L = \sum_{ij} r_{ij} x_{ij} \quad \text{,s.t.} \sum_{j} x_{ij} = 1 \text{ and } \; x_{ij} \in \{0, 1\}$$
    to obtain a matching between states in G_{L-1} and states in G_L.
5. If $|m_L - m_{L-1}| = 0$, then stop. The current value of L is the correct value.

Bibliography

[1] Beaver - a LALR Parser Generator. `http://beaver.sourceforge.net/`.

[2] DSL information. `http://www.damnsmalllinux.org`.

[3] How Win32/Zbot Works. `http://www.microsoft.com/security/sir/story/default.aspx#\protect\kern-.1667em\relaxzbot_works`.

[4] The banking malware scourge. `http://searchsecurity.techtarget.com/magazineContent/The-banking-malware-scourge`.

[5] The threat from P2P botnets. `http://www.securelist.com/en/blog/654/Lab_Matters_The_threat_from_P2P_botnets`.

[6] Tor project. `http://www.torproject.org/`.

[7] Tshark. `http://www.wireshark.org/docs/man-pages/tshark.html`.

[8] UAB computer forensics links internet postcards to virus. `http://www.hindu.com/thehindu/holnus/008200907271321.htm`.

[9] ZeuS Gets More Sophisticated Using P2P Techniques. `http://www.abuse.ch/?p=3499`.

[10] Zeus Tracker. `https://zeustracker.abuse.ch`.

[11] *Applied Regression Analysis and Other Multivariable Methods.* Duxbery Press, 1998.

[12] C. Abbott. Strengthening the Anonymity of Anonymous Communication Systems. Master's thesis, Clemson University, 2011.

[13] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools, 2nd Edition.* Pearson Education Inc, 2006.

[14] M. A. Aycinena. Probabilistic Geometric Grammars for Object Recognition. Master's thesis, Massachusetts Institute of Technology, 2005.

[15] J. C. Willems B. D. Moor B. Vanluyten. Equivalence of state representations for hidden Markov models. *Systems & Control Letters*, 57:410–419, 2008.

[16] J. Bennett and S. Lanning. The Netflix prize. In *Proc. KDDCup '07*, 2007.

[17] T. Pfister B. Beutler, R. Kaufmann. Integrating a non-probabilistic grammar into large vocabulary continuous speech recognition. In *2005 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 104–109, 2005.

[18] H. Bhanu, J. Schwier, R. Craven, R. Brooks, K. Hempstalk, D. Gunetti, and C. Griffin. Side-Channel Analysis for Detecting Protocol Tunneling. *Advances in Internet of Things*, 1(2):13–26, 2011.

[19] Harikrishnan Bhanu. Timing Side-Channel Attacks On SSH. Master's thesis, Clemson University.

[20] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '07)*, pages 37–48, 2007.

[21] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho. Seven Years and One Day: Sketching the Evolution of Internet Traffic. In *INFOCOM 2009, IEEE*, pages 711 –719, april 2009.

[22] R. Brooks, J. Schwier, and C. Griffin. Behavior detection using confidence intervals of hidden Markov models. *IEEE Trans. on Systems, Man, and Cybernetics, part B*, 39(6):1484–1492, December 2009.

[23] Christopher D. Brown and Herbert T. Davis. Receiver operating characteristics curves and related decision measures: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 80(1):24 – 38, 2006.

[24] Z. Chi. Statistical Properties of Probabilistic Context-Free Grammars. *Computational Linguistics*, 25(1):131–160, 1999.

[25] Z. Chi and S. Geman. Estimation of Probabilistic Context-Free Grammars. *Computational Linguistics*, 24(2):299–305, 1998.

[26] Chia Yuan Cho, Domagoj Babi ć, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 426–439, New York, NY, USA, 2010. ACM.

[27] A. Corazza and G. Satta. Probabilistic Context-Free Grammars Estimated from Inifinte Distributions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(8):1379–1393, 2007.

[28] Ryan Michael Craven. Traffic Analysis of Anonymity Systems. Master's thesis, Clemson University.

[29] F. Dennis. Experts predict Storm Trojan's reign to continue. `http://searchsecurity.techtarget.com/news/1278241/Experts-predict-Storm-Trojans-reign-to-continue`.

[30] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the* 13^{rd} *USENIX Security Symposium*, pages 303–320, 2004.

[31] R. Durbin, S. Eddy, A. Krogh, and G. Mitchinson. *Biological sequence analysis : probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[32] M. Falkhausen, H. Reininger, and D. Wolf. Calculation of Distance Measures Between Hidden Markov Models. In *Proc. Eurospeech*, pages 1487–1490.

[33] FBI. Cyber Banking Fraud. `http://www.fbi.gov/news/stories/2010/october/cyber-banking-fraud`.

[34] G. S. Fishman. A First Course in Monte Carlo. *Duxbury Advanced Series*, 2006.

[35] X. Fu, B. Graham, R. Bettati, and W. Zhao. On effectiveness of link padding for statistical traffic analysis attacks. In *Proc. 23rd IEEE Conference on Distributed Computing Systems*, pages 340–349, 2003.

[36] X. Fu, B. Graham, R. Bettati, W. Zhao, and D. Xuan. Analytical and empirical analysis of countermeasures to traffic analysis attacks. In *Proc. 32nd International Conference on Parallel Processing*, pages 483–492, 2003.

[37] S. Geman and M. Johnson. Probabilistic grammars and their applications. *In International Encyclopedia of the Social & Behavioral Sciences*, pages 12075–12082, 2002.

[38] D. George. Storm Botnet storms the Net. http://ieet.org/index.php/IEET/more/dvorsky20070927/.

[39] S. C. Geyik. *Network Data Modeling Via Grammatical Structures.* PhD thesis, Rensselaer Polytechnic Institute, 2012.

[40] Zoubin Ghahramani. Learning dynamic Bayesian networks. In *Adaptive Processing of Sequences and Data Structures*, pages 168–197. Springer-Verlag, 1998.

[41] C. Griffin, R. Brooks, and J. Schwier. A Hybrid Statistical Technique for Modeling Recurrent Tracks in a Compact Set. *IEEE Trans. Automatic Control*, 2011.

[42] J.L. Gross and J. Yellen. *Handbook of Graph Theory.* CRC Press, 2004.

[43] G Gu. *Correlation-Based Botnet Detection In Enterprise Networks.* PhD thesis, College of Computing, Georgia Institute of Technology, 2008.

[44] G. Gu, J. Zhang, and W. Lee. BotSniffer - Detecting Botnet Command and Control Channels in Network Traffic. *15th Annual Network and Distributed System Security Symposium*, 2008.

[45] F. Hernández-Campos, J. S. Marron, Gennady Samorodnitsky, and F. D. Smith. Variable heavy tails in internet traffic. *Perform. Eval.*, 58(2+3):261–261, November 2004.

[46] J. Hopcroft and J. Ullman. *Formal languages and their relation to automata.* Addison-Wesley Longman Publishing Co. Inc., 1969.

[47] Y. A. Ivanov. Application of Stochastic Grammars to Understanding Actions. Master's thesis, Massachusetts Institute of Technology, 1992.

[48] B. H. Juang and L. R. Rabiner. A Probabilistic Distance Measure for Hidden Markov Models. *ATT Technical Journal*, 64(2):391–408, 1985.

[49] S. Kevin. Worm 'Storm' gathers strength. http://www.neoseeker.com/news/7103-worm-storm-gathers-strength/.

[50] Roger E. Kirk. *Statistics: An Introduction.* Wadsworth Publishing, 2007.

[51] K. Lari and S. J. Young. The Estimation of Stochastic Context-Free Grammars using the Inside-Outside Algorithm. *Computer Speech and Language*, 4(1):35–56, 1990.

[52] K. Lari and S. J. Young. Applications of Stochastic Context-Free Grammars using the Inside-Outside Algorithm. *Computer Speech and Language*, 5(3):237–257, 1991.

[53] Tuneesh Lella and Riccardo Bettati. Privacy on encrypted voice-over-ip. In *Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC-2007)*, 2007.

[54] Chen Lu and Richard Brooks. Botnet traffic detection using hidden Markov models. In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '11, pages 31:1–31:1, New York, NY, USA, 2011. ACM.

[55] Chen Lu and Richard Brooks. Botnet traffic detection using hidden Markov models. *IEEE Trans. Dependable Secur. Comput.*, Under review.

[56] Chen Lu and Richard R. Brooks. P2P Hierarchical Botnet Traffic Detection Using Hidden Markov Models. In *Learning from Authoritative Security Experiment Results (LASER) Workshop*, 2012.

[57] Chen Lu, Jason Schwier, Ryan Craven, Lu Yu, Richard Brooks, , and Chris Griffin. A Normalized Statistical Metric Space for Hidden Markov Models. *IEEE Trans. on Systems, Man, and Cybernetics, part B*, In Press.

[58] C. D. Manning and H. Schutze. Foundations of Statistical Natural Language Processing. *The MIT Press*, 1999.

[59] Tamara Mihaljev, Lucilla de Arcangelis, and Hans J. Herrmann. Inter-arrival times of message propagation on directed networks, 2012.

[60] Michael I. Miller and Joseph A. O'Sullivan. Entropies and Combinatorics of Random Branching Processes and Context-Free Languages. *IEEE Trans. on Information Theory*, 38(4):1292–1310, 1992.

[61] S. Mitra and T. Acharya. Gesture Recognition: A Survey. *IEEE Trans. on Systems, Man, and Cybernetics, part C*, 37(3):311–324, 2007.

[62] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, pages 31–88, 2001.

[63] J. Neter, W. Wasserman, and M. H. Kutner. Applied linear Regression models. *Irwin Press*, 1989.

[64] Chomsky Noam. Three models for the description of language. *Information Theory, IRE Transactions*, September 1956.

[65] Christopher Null. Scary 'global hacking offensive' finally outed. Yahoo! Tech. Retrieved February 23, 2010.

[66] G. Ollmann. Botnet Communication Topologies. *White Paper of Damballa*, 2009.

[67] Roberto Perdisci, Igino Corona, and Giorgio Giacinto. Early Detection of Malicious Flux Networks via Large-Scale Passive DNS Traffic Analysis. *IEEE Trans. Dependable Secur. Comput.*, 9(5), 2012.

[68] W. H. Press, S. A. Teukosky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd Edition.* Cambridge University press, 1992.

[69] K. Pulasinghe, K. Watanabe, K. Izumi, and K. Kiguchi. Modular Fuzzy-neuro Controller Driven by Spoken Language Commands. *IEEE Trans. on Systems, Man, and Cybernetics, part B*, 34(1):293–302, Feburary 2004.

[70] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, Feburary 1989.

[71] S. Sahraeian and B. Yoon. A Novel Low-Complexity HMM Similarity Measure. *Signal Processing Letters, IEEE*, 18(2):87–90, 2011.

[72] R.L. Scheaffer. *Introduction to Probability and Its Applications, Second Edition.* 1995.

[73] Antoine Scherrer, Nicolas Larrieu, Philippe Owezarski, Pierre Borgnat, and Patrice Abry. Non-Gaussian and Long Memory Statistical Characterizations for Internet Traffic with Anomalies. *IEEE Trans. Dependable Secur. Comput.*, 4(1):56–70, January 2007.

[74] R.E. Schumacker and R.G. Lomax. *A Beginners Guide to Structural Equation Modeling.* Lawrence Erlbaum Associates, Inc., 2004.

[75] J. Schwier. *Pattern Recognition for Command and Control Data Systems.* PhD thesis, Clemson University, 2009.

[76] J. M. Schwier, R. R. Brooks, C. Griffin, and S. Bukkapatnam. Zero knowledge hidden Markov model inference. *Pattern Recognition Letters*, 30(14):1273–1280, 2009.

[77] C.R. Shalizi. *Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata.* PhD thesis, University of Wisconsin Madison, May 2001.

[78] C.R. Shalizi and J.P. Crutchfield. Computational Mechanics: Patterns and Prediction, Structure and Simplicity. Technical report, Santa Fe Institute, 2001.

[79] C.R. Shalizi and K.L. Shalizi. Blind Construction of Optimal Nonlinear Recursive Predictors for Discrete Sequences. *arXiv:cs.LG/0406011 v1*, June 2004.

[80] C.R. Shalizi, K.L. Shalizi, and J.P. Crutchfield. An algorithm for pattern discovery in time series. *arXiv:cs.LG/0210025 v3*, November 2002.

[81] J. Silva and S. Narayanan. Upper Bound Kullback-Leibler Divergence for Transient Hidden Markov Models. *IEEE Transactions on Signal Processing*, pages 4176–4188, 2008.

[82] B. K. Sin, J. Y. Ha, S. C Oh, and J. H. Kim. Network-based approach to online cursive script recognition. *IEEE Trans. on Systems, Man, and Cybernetics, part B*, 29(2):321–328, April 1999.

[83] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th conference on USENIX Security Symposium*, 2001.

[84] T. Starner and A. Pentland. Visual recognition of American sign language using hidden Markov models. *Perceptual Computing Section, the Media Laboratory, Massachusetts Institute of Technology, Tech. Rep.*, 1995.

[85] Tara L. Steuber, Peter C. Kiessler, and Robert Lund. Testing For Reversibility in Markov Chain Data. *Probability in the Engineering and Informational Sciences*, 26(04):593–611, 2012.

[86] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 635–647, 2009.

[87] S. Stover, D. Dittrich, J. Hernandez, and S.Dietrich. Analysis of the Storm and Nugache trojans: P2P is here. *USENIX*, 32(6), 2007.

[88] W. Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet Detection Based on Network Behavior. *Advances in Information Security, Springer*, pages 1–24, 2008.

[89] Dyrka W. Probabilistic Context-Free Grammar for pattern detection in protein sequences, 2007.

[90] Dyrka W. and Nebel J. C. A stochastic context free grammar based framework for analysis of protein sequences. *BMC Bioinformatics*, 10(323), 2009.

[91] R. Walpole and R. H. Myers. *Probability and Statistics for Engineers and the Scientists.* Collier Macmillan, 1989.

[92] Xinyuan Wang, Douglas S. Reeves, and S. Felix Wu. Inter-Packet Delay Based Correlation for Tracing Encrypted Connections through Stepping Stones. In *Proceedings of the 7th European Symposium on Research in Computer Security, ESORICS '02*, pages 244–263, 2002.

[93] Lu Wei, Tavallaee Mahbod, and Ali A. Ghorbani. Automatic discovery of botnet communities on large-scale communication networks. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 1–10, New York, NY, USA, 2009. ACM.

[94] E. W. Weisstein. *CRC Concise Encyclopedia of Mathematics.* Chapman and Hall/CRC, 1999.

[95] S. Wellek. *Testing statistical hypotheses of equivalence.* Chapman and Hall/CRC, 2003.

[96] Clay Wilson. Botnets, Cybercrime, and Cyberterrorism: Vulnerabilities and Policy Issues for Congress. *CRS Report for Congress*, 2009.

[97] L. Yu, J. Schwier, R. Craven, R. Brooks, and C. Griffin. Inferring statistically significant hidden Markov models. *IEEE Trans. on Knowledge and Data Engineering*, In press.

[98] Wei Yu, Xun Wang, Prasad Calyam, Dong Xuan, and Wei Zhao. Modeling and Detection of Camouflaging Worm. *IEEE Trans. Dependable Secur. Comput.*, 8(3), 2011.

[99] J. Zeng, J. Duan, and C. Wu. A New Distance Measure for Hidden Markov Models. *Exp. Syst. Applicat.*, 37(2):1550–1555, 2010.

[100] S. Zhong. *Probabilistic Model-based Clustering of Complex Data.* PhD thesis, The University of Texas at Austin, 2003.

[101] D. Zwillinger. Standard mathematical tables and formulae. *Chapman & Hall/CRC*, 2003.