UNIVERSITY OF CALIFORNIA
RIVERSIDE

A Continuous Time Bayesian Network Approach for Intrusion Detection

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Jing Xu

August 2010

Dissertation Committee:
Dr. Christian R. Shelton, Chairperson
Dr. Michalis Faloutsos
Dr. Eamonn Keogh

The Dissertation of Jing Xu is approved:

Committee Chairperson

University of California, Riverside

Acknowledgements

It would have been next to impossible to write this thesis without the help and guidance from my esteemed advisor, Dr. Christian R. Shelton. I would not be able to even start my PhD study in the field of artificial intelligence without his encouragement. Ever since I joined his research group, I have received tremendous advice and help from him. Whenever I get stuck and feel frustrated on challenging periods, he is always there to support me and guide me to the right direction. His enthusiasm and sharp insights into the research give me a great amount of inspiration. His helpful supervision from the preliminary to the concluding level enables me to achieve many progresses. He is one of those persons that I may owe my deepest gratitude to for my lifetime. I am heartily thankful to you, Christian!

My great thanks also go to my committee members, Dr. Michalis Faloutsos and Dr. Eamonn Keogh. They have given me many great suggestions during my research. Thank you for your help on making this thesis better!

I would also thank my lab mates in RLAIR, Dr. Yu Fan, William Lam and Joon Lee. I will always remember the time when we collaborated on the CTBN code base. The trust, support, communication between us help me overcome a lot of difficulties and make this teamwork successful and enjoyable. I would also thank other group members, Dr. Teddy Yap and Antony Lam for all the interesting conversations and discussions that bring me so much happiness.

My last, and deepest, acknowledgement goes to my dear husband, the big Daddy of my beloved daughter April, my true love, Dr. Guobiao Mei. He has always been on my side,

supporting me and loving me. As a former lab mate, he also gives me numerous valuable suggestions and help. Guobiao, I feel so lucky to have met you and married you! I would also thank my parents and my parents-in-law, who give me so much faith and love. I offer my best regards and blessings to all of you.

ABSTRACT OF THE DISSERTATION

A Continuous Time Bayesian Network Approach for Intrusion Detection

by

Jing Xu

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, August 2010
Dr. Christian R. Shelton, Chairperson

Network attacks on computers have become a fact of life for network administrators. Detecting attacks accurately is important to limit their scope and destruction. Intrusion detection systems (IDSs) fall into two high-level categories: network-based systems (NIDS) that monitor network behaviors, and host-based systems (HIDS) that monitor system calls. In this work, we present a general technique for both systems.

We consider the problem of detecting intrusions of the host level. We use anomaly detection, which identifies patterns not conforming to a historic norm. Our approach does not require expensive labeling or prior exposure to the attack type. In both types of systems, the rates of change vary dramatically over time (due to burstiness) and over components (due to service difference). To efficiently model such systems, we use continuous time Bayesian networks (CTBNs) and avoid specifying a fixed time interval. We build generative models from historic non-attack data, and flag future event sequences whose likelihood under this norm is below a threshold.

As a NIDS, our method differs from previous approaches in explicitly modeling temporal dependencies in the network traffic. Our models are therefore more sensitive to subtle variations in the sequences of network events. We first construct a factored CTBN model for the network packet traces. We present two simple extensions to CTBNs that allow for instantaneous events that do not result in state changes, and simultaneous transitions of two variables. We then extend this model to a connected one. We construct it in a hierarchical way and use Rao-Blackwellized particle filtering for inference. We illustrate the power of our method through experiments on detecting real worms and identifying hosts on two publicly available network traces, the MAWI dataset and the LBNL dataset.

For HIDS, we develop a novel learning method to deal with the finite resolution of system log file time stamps, without losing the benefits of our continuous time model. We demonstrate the method by detecting intrusions in the DARPA 1998 BSM dataset.

Contents

List of Figures

Chapter 1

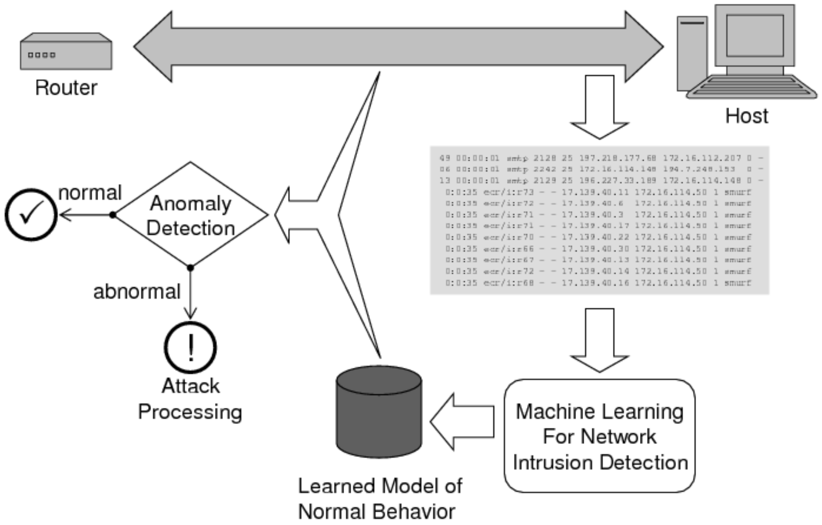
Introduction

As a result of the outstanding growth of the Internet, the world now has become a "global village." Its fast speed, wide coverage and easy access provide a powerful platform for governments, business and individuals to communicate or share the information. Internet users could make great use out of the rich source of a world wide network. However, they also expose their vulnerability to being attacked. In addition, misuse or abuse of one user's computer also challenges network administrators to deal with identity theft. These attempts that try to invade a system and compromise the availability or quality of the system services are often called network intrusions. Network intrusions can be harmful or even fatal to the stability and security of information systems. They can cripple the network and compromise the confidentiality of personal or economic information. Our goal is to detect such attacks. Detecting attacks quickly is important in limiting their scope and destruction.

1.1 Intrusion Detection Systems

A system that can detect network intrusions automatically is called an Intrusion Detection System (IDS). In our work, we approach the problem from the point of view of anomaly detection, which identifies patterns not conforming to a historic norm. In the machine learning field, it is often approached with unsupervised learning. The detector we build is at the host level. Instead of constructing a detector for the network as a whole, our goal is to detect these attacks that attempt to compromise the performance quality of a particular host machine. Our method can be employed at each computer separately to determine whether a particular host has been under attack. While we lose global information, we gain speed, individual tuning, and robustness. A network under attack may not be able to aggregate information to a central detector.

Figure 1.1 shows an overview of our host level network attack detection system. We only focus on a single computer (a host) on the network. We use unsupervised learning to build a model of the normal behavior of this host-level on its network traffic, without looking into its system behavioral state like resource usage and file access. Those activities that differ from this norm are flagged as possible intrusion attempts. Our host-based intrusion detection system works in a similar fashion, except that the model is built on system calls.



Router
Host
normal
Anomaly Detection
abnormal
Attack Processing
Learned Model of Normal Behavior
Machine Learning For Network Intrusion Detection

```
49 00:00:01 smtp 2128 25 197.218.177.68 172.16.112.207 0 -
06 00:00:01 smtp 2242 25 172.16.114.148 194.7.248.153  0 -
13 00:00:01 smtp 2129 25 196.227.33.189 172.16.114.148 0 -
 0:0:35 ecr/i:r73 - - 17.139.40.11 172.16.114.50 1 smurf
 0:0:35 ecr/i:r72 - - 17.139.40.6  172.16.114.50 1 smurf
 0:0:35 ecr/i:r71 - - 17.139.40.3  172.16.114.50 1 smurf
 0:0:35 ecr/i:r71 - - 17.139.40.17 172.16.114.50 1 smurf
 0:0:35 ecr/i:r70 - - 17.139.40.22 172.16.114.50 1 smurf
 0:0:35 ecr/i:r66 - - 17.139.40.30 172.16.114.50 1 smurf
 0:0:35 ecr/i:r67 - - 17.139.40.13 172.16.114.50 1 smurf
 0:0:35 ecr/i:r72 - - 17.139.40.14 172.16.114.50 1 smurf
 0:0:35 ecr/i:r68 - - 17.139.40.16 172.16.114.50 1 smurf
```

Figure 1.1: The architecture of the intrusion detection system. The model is learned from normal traces. Anomaly detection compares the model to current network traffic. The output is the predicted labels for each time window.

1.2 Continuous Time Bayesian Networks Approach

Network traffic traces and system call logs are two common sources of audit data that facilitate intrusion detections at the host level. They both log activities associated with a computer. In a network-based intrusion detection system (NIDS), the network packet traces are monitored. Network traffic traces collect information from a network's data stream and provide an external view of the network behaviors. In a host-based intrusion detection system (HIDS), the internal state of a computing system is analyzed. System call logs monitor executing programs' operating system calls. These activities usually evolve over time. So a static model that does not account for the element of time is usually outperformed by a dynamic one. In both types of systems, the rates of change vary dramatically over time (due to burstiness) and over components (due to service differences). A computer user might be alternatingly busy and resting. During the busy period, bursts of action often happen causing a peak of network traffic flow or operating system usage during a very short time. However, during the resting period, the computer just maintains its regular running pattern, and network or system activities are much less intense, e.g. automatically checking email every few minutes. Thus, a dynamic model that requires discretizing the time, i.e. a Dynamic Bayesian Network (DBN), is not efficient or even feasible. Such models propagate information at fixed time steps. This means even if some of the components in the system have slow-paced events, the entire system still have to run at the finest time granularity. We develop intrusion detection techniques using continuous time Bayesian networks (CTBNs) (Nodelman et al., 2002) for both data types (network and internal state). CTBNs are factored representations of continuous time Markov processes. They describe the temporally evolving dynamics of stochastic variables. CTBNs have been successful in other applications (Ng et al., 2005; Gopalratnam et al., 2005), but have not previously been used for detecting network intrusions. Although NIDS and HIDS are of different formats and points of interest, we demonstrate the flexibility of a CTBN to describe either.

1.2.1 Network-based Intrusion Detection Systems

Our first effort is to detect anomalies from network traffic traces (NIDS). Our approach differs from previous approaches in a number of key ways. It is adaptive and constructed at the host level. It does not treat the packets or connections as *i.i.d.* sequences, but respects the ordering of the network traffic. Finally, it does not model the traffic features as normal or exponential distributions. Many features of network traffic are distinctly non-Gaussian and often multimodal.

While anomalies may be very subtle and difficult to detect, the more subtle the attack, the longer the attack will take and the more it will stress the patience of the attacker. Looking at summarized information like flow statistics is not helpful especially for stealthy worms which can mingle well with normal traffic by sacrificing their spreading speed and scale. We, therefore, feel that looking for abnormalities in the detailed network traffic flow level is a utile method for finding attacks. A network flow for a given host machine is a sequence of continuous-time asynchronous events. Furthermore, these events form a complex structured system, where statistical dependencies relate network activities like packet emissions and connection starts. We employ a generative probabilistic model to describe such dynamic processes evolving over continuous time. In particular, we use CTBNs to reason about these structured stochastic network processes.

One successful detector we build employs a completely factored CTBN model. We make the assumption that network services associated with different destination ports are independent of each other. We model each port's traffic with its own CTBN submodel. Since the whole model is fully factored, we can estimate the model parameters for each submodel individually. We use the exact inference algorithm described by Nodelman et al. (2002) to learn the parameters. Once the CTBN model has been fit to historic data, we detect attacks by computing the likelihood of a window of the data under the model. If the likelihood falls below a threshold, we flag the window as anomalous. Otherwise, we mark it as normal.

A second successful CTBN detector we build uses a connected model. To allow our model to be more descriptive, we remove the restriction that port-level submodels are independent by introducing another latent variable that ties the submodels together. The exact inference algorithm is no longer tractable for learning the parameters for this connected model. We use a Rao-Blackwellized particle filtering (RBPF) method to approximately learn the parameters. For testing, we use the same technique as above: calculating the likelihood of a window of the data under the model, and comparing it to a predefined threshold.

We compared the performance of the detectors that use the factored model and the fully connected model on real network traffic. The datasets we used are the MAWI working group backbone traffic (MAWI) and the LBNL/ICSI internal enterprise traffic (LBNL).

We also approach the problem of speeding up the necessary calculations in CTBN reasoning. Although the above methods achieve good performance on real data, the overhead in the detection phase is not ignorable. We train the models offline and beforehand. Thus long training times are acceptable. But instant feedback is expected upon deployment. The sooner the alert triggers, the less the scope of the destruction is. In the detection phase, the likelihood of a time window under the learned model is calculated. This involves the exact inference in a CTBN and usually takes a large amount of time. We present an approximation method to accelerate the reasoning.

1.2.2 Host-based Intrusion Detection Systems

Our second effort is to detect intrusions using system call logs (HIDS). A system log file contains an ordered list of calls made to a computer's operating system by a program. Research focuses on analyzing the ordering and the context of the sequence, rather than simply counting the overall statistics. A CTBN is a natural way of modeling such sequential data.

The CTBN model is similar to our port-level network model. Individual system calls, which are the event description fields in the header token, are transiently observed: they happen instantaneously with no duration. We also introduce a hidden variable to allow correlations among system calls. This hidden variable models the internal state of the machine to some extent.

Because of the finite resolution of the computer's clock, all the system calls issued within a clock tick are assigned the same time stamp. Therefore the data stream consists of long periods of time with no activity, followed by sequences of calls in which the order is correctly recorded, but the exact timing information is lost. This poses a new challenge for CTBN reasoning. We present a learning method for such data without resorting to time discretization. When testing, the likelihood of a whole process is reported as the score and compared to a threshold, to determine if the process is an attack.

1.3 Disertation Contributions

Our work is one of the few that provide a general solution for both NIDS and HIDS. We are the first method to model network traffic traces and system call logs using a continuous time model. And, our work is among the first approaches that apply CTBNs to real world applications. We also contribute to theoretic research in CTBNs by introducing toggle variables, a Rao-Blackwellized particle filtering inference method, and a parameter learning algorithm for observations within finite resolution clocks. Our work achieve good performance on real datasets.

1.4 Outline

In this dissertation, we present a CTBN approach for intrusion detection for both network-based data and host-based data. The dissertation is organized in the following structure.

In Chapter 2, we review the problem and background of intrusion detection.

In Chapter 3, we give a brief background of CTBNs.

In Chapter 4, we describe our CTBN approach for NIDS.

In Chapter 5, we show our experiment results for NIDS.

In Chapter 6, we present our speed up technique and some other work.

In Chapter 7, we describe our CTBN approach for HIDS.

In Chapter 8, we discuss the contributions of this dissertation from the realm of both CTBN reasoning and intrusion detection.

Chapter 2

Intrusion Detection Problem

In this chapter, we review the problem of intrusion detection and discuss the related work in this field.

2.1 Intrusion Detection System

In the field of information security, researchers have devoted themselves to the problem of intrusion detection for many decades. Generally speaking, intrusions can be detected manually or automatically. While human intervention can improve security, it comes at the cost of increased time and effort. Humans cannot analyze each and every event to determine if it might be part of an attack, so there is a need for a method that can adapt to the role and usage patterns of the system, while still detecting attacks automatically to perform such checks. A system that can detect intrusion automatically is called an intrusion detection systems (IDS).

Administrators can use hardware devices, i.e. security cameras or motion sensors, to observe the physical status of the system. These devices are usually called physical IDS. They are often used as intrusion prevention systems as well. People can also install software to monitor the event logs of the system. Those that monitor the flow of network packets are called network-based intrusion detection systems (NIDS), e.g. SNORT. Those that examine the system calls are called host-based intrusion detection systems (HIDS), e.g. OSSEC. NIDS and HIDS are used for identifying attacks. In this dissertation, we focus on the software intrusion detection systems, including both NIDS and HIDS.

An IDS can also be categorized into two types: misuse detection systems and anomaly detection systems. Misuse detection systems build a database of all the malicious signatures seen previously, and compare the system behavior against each of them. If the behavior matches any of the recorded bad signature, it is flagged as an known type of attack, not only an intrusion. It is usually approached by the method of a supervised learning algorithm in the machine learning field. Supervised learning has a training data set, which is composed of pairs of a data example and its output value. Its task is to predict an output for a given data example. When the output is a continuous value, it is called a *regression* task; when the output is a class label, it is called a *classification* task. In the case of misuse detection, a label (an attack type) is predicted. Therefore, it is a typical classification problem and can be approached by many existing algorithms. However, it has the disadvantage that it requires updating the database whenever a new attack signature is encountered. Attacks vary greatly and new types of attacks are invented frequently. The rapid development of new viruses makes maintaining such a database time-consuming and error-prone. Therefore, a supervised learning method that attempts to distinguish good from bad network traffic based on historic labeled data is necessarily limited in its scope.

Anomaly detection systems work just in the opposite manner. They assume a bottom line for the normal pattern, and compare the behavior against this norm. Unlike in misuse detection systems where labeled data of "bad" behavior are collected, gathering normal or good network traffic or system calls is relatively simple. It is often possible to designate times when we can be reasonably certain that no attacks occurred and use all of the data during that span. For an attack to be successful, it must differ in some way from normal network traffic. Anomaly detection can identify new attacks even if the attack type was unknown beforehand. Unsupervised learning, which is given only unlabeled data, allows the anomaly detector to adapt to changing environments, thereby extending its domain of usefulness. By modeling normal behavior from historic clean data, we can identify abnormal activity without a direct prior model of the attack by simply comparing its deviation from the learned norm. Our method falls into this category. However, anomaly detection has the disadvantage that it can not classify attack types. Also, it is not as accurate as misuse detection where a large database of seen attacks are maintained. It may have higher false alarm where some normal behavior is misidentified as abnormal just because it is different from the "usual" normal pattern.

2.2 Literature Study

Much of the previous work in intrusion detection focuses on one area only — either detecting the network traffic or mining the system call logs. The role taken by Eskin et al. (2002) is similar to our approach in that they apply their method to both of these kinds of data. They map data elements to a feature space and detect anomalies by determining which points lie in sparse regions using cluster-based estimation, K-nearest neighbors and one-class SVM. They use a data-dependent normalization feature map for network traffic data and a spectrum kernel for system call traces. We are different in that we model the evolving dynamics of both of the data types, instead of extracting static features. We compare with their algorithm that applies one-class SVM on spectrum string kernel in our experiments on both of the data type. Lee and Stolfo (1998) provided another example of developing a general solution to both network tcpdump and system call data. But unlike our work, they built classifiers to detect known intrusions.

2.2.1 NIDS

In the area of NIDS, most of the previous work approaches the detection problem from the viewpoint of either misuse detection or anomaly detection. We review the related work using both of the techniques.

Misuse Detection

Many learning, or adaptive, methods have been proposed for network data. As described in Section 2.1, misuse intrusion detection techniques often build a classifier on the network data. If an attack has appeared before in their training data, it can be easily recognized in the future.

As a signature-based detection algorithm, Karagiannis et al. (2005) proposed a traffic classification technique named "BLINC." They classify traffic flows at multiple levels according to the applications that generate them. We share many of the assumptions of their work. In particular, we also assume that we do not have access to the internals of the machines on the networks. We differ in that our approach does not rely on human intervention and interpretation, nor do we assume that we have access to network-wide traffic information. Network-wide data and human intervention have advantages, but they can also lead to difficulties (data collation in the face of an attack and increased human effort), so we chose to leave them out of our solution. Zuev and Moore (2005) like our approach, model traffic with graphical models, in particular Naive Bayes networks. But their goal is to categorize network traffic based on applications instead of detecting attacks. Soule et al. (2004) also approach the problem as a classification task using histograms. They are very similar in statistical flavor to our work. They also fit a distribution (in their case, a histogram modeled as a Dirichlet distribution) to network data. However, they model flow-level statistics, whereas we work at the level of individual connections. Additionally, they attempt network-wide clustering of flows instead of anomaly detection. Dewaele et al. (2007) profile the statistical characteristics of anomalies by using random projection techniques (sketches) to reduce the data dimensionality and a multi-resolution non-Gaussian marginal distribution to extract anomalies at different aggregation levels. Kruegel et al. (2003) present a Bayesian approach to the detecting problem as an event classification task while we only care about whether the host is under attack during an interval. In addition, they also use system monitoring states to build their model, which we do not employ.

The goal of such papers is usually not to detect attacks but rather to classify non-attacks by traffic type; if applied to attack detection, they would risk missing new types of attacks. Furthermore, they frequently treat each network activity separately, instead of considering their temporal context.

Anomaly Detection

Anomaly detection is another popular method for intrusion detection. It usually employs data partition (or clustering). It does not require labeled data and can detect attacks not seen before.

Lakhina et al. (2005) have a nice summary of adaptive (or statistical) methods that look at anomaly detection (instead of classification). They use an entropy-based method for the entire network traffic. Many of the other methods, such as that of Ye et al. (2002), use either statistical tests or subspace methods that assume the features of the connections or packets are normally distributed. This is not always true. For instance, the distribution of the number of connections at any time on a machine is asymmetric. The probability of this number being small is fairly large. It could be simply because the computer user is not active. However, the probability of this number being high or even exceeding a threshold is small (which could be an indication of an attack). Rieck and Laskov (2007) model the language features like n-grams and words from connection payloads. Xu et al. (2005) also uses unsupervised methods, but they concentrate on clustering traffic across a whole network. Similarly, Soule et al. (2005) build an anomaly detector based on Markov models, but it is for the network traffic patterns as a whole and does not function at the host level.

Lazarevic et al. (2003) is similar to our work. It is one of the few papers to attempt to find attacks at the host level. They employ nearest neighbor, a Mahalanobis distance approach, and a density-based local outliers method, each using 23 features of the connections. Their methods make the standard *i.i.d.* assumption about the data (and therefore miss the temporal context of the connection) and use 23 features (compared to our few features). Agosta et al. (2007b) present an adaptive detector whose threshold is time-varying. It is similar to our work in that they also rely on model-based algorithms. But besides the usage of network signature data, they look at host internal states like CPU loads which are not available to us.

While there has been a great variety of previous work, our work is novel in that it detects anomalies at the host level using only the timing features of network activities. We do not consider each connection (or packet) in isolation, but rather in a complex context. We capture the statistical dynamic dependencies between packets and connections to find sequences of network traffic that are anomalous *as a group*.

2.2.2 HIDS

Previous work on detecting intrusions in system call logs can be roughly grouped into two categories: sequence-based and feature-based. Sequence-based methods focus on the sequential order of the events while feature-based methods treat system calls as independent data elements. Our method belongs to the former category since we use a generative model to describe the dynamics of the sequences.

Time-delay embedding (tide) and sequence time-delay embedding (stide) are two examples of sequence based methods (Forrest et al., 1996; A.Hofmeya et al., 1998). They generalize the data by building a database storing previously seen system call sub-sequences, and test by looking up subsequences in the database. These methods are straightforward and often achieve good results. We compare with them in our experiments. Tandon and Chan (2005) look at a richer set of attributes like the return values and the arguments associated with a system call, while we only make use of the system call names. Cha (2005) uses the Soundex algorithm to change variable length sequential system call data into a fixed length behavior pattern.

Feature-based methods like those of Hu et al. (2003) use the same dataset we use, the DARPA 1998 BSM dataset, but their training data is noisy and they try to find a classification hyperplane using robust support vector machines (RSVMs) to separate normal system call profiles from intrusive ones. Eskin (2000) also works on noisy data. They make the assumption that their training data contains a large portion of normal elements and few anomalies. They present a mixture of distribution over normal and abnormal data and calculate the likelihood change if a data point is moved from normal part to abnormal part to get the optimum data partition.

Yeung and Ding (2002) try to use both techniques. They provide both dynamic and static behavioral models for system call data. For the dynamic method, a hidden Markov model (HMM) is used to model the normal system events and a likelihood is calculated for each testing sequence and compared against a certain threshold. Our work for the system call traces problem is very close to their framework since we also build a dynamic model for the sequential data and compute the likelihood of a testing example as a score. But we are different in that our CTBN models the continuous-time dynamics rather than time-sliced behaviors. For the static method, they represent the normal behavior by a command occurrence frequency distribution and measure the distance from the testing example to this norm by cross entropy. The dataset they use is KDD archive dataset.

2.3 Dynamic Process Approach

Activities on a computer as a whole can be viewed as a single process whose state changes from one to another. A dynamic model describes the state transitions of a process along time. If we allow uncertainty in its evolution, the process becomes a stochastic one. Probabilistic models are often used to model such processes. Researchers have provided quite a few dynamic process approaches in the field of intrusion detection. As mentioned in Section 2.2.2, Yeung and Ding (2002) use a Hidden Markov Model (HMM) to model the system calls.

On the other hand, computer activities contain sequences of events. For instance, a network flow is composed of traces of packet transmission and receipts, and connection establishments and terminations. The process is an integrated whole system formed by these events as components. The sequence and timing of events are very important in network traffic flow. It matters not just how many connections were initiated in the past minute, but also their timing: If they were evenly spaced the trace is probably normal, but if they all came in a quick burst it is more suspecious. Similarly, the sequence is important. If the connections were made to sequentially increasing ports it is more likely to be a scanning virus, whereas the same set of ports in random order is more likely to be normal traffic. These are merely simple examples. We would like to detect more complex patterns. In most cases, the emission of such events are not in isolation, but in complex context. There can be interactive relations between them. Therefore, a graphical model that captures the structural dependencies among the elements of a network is well suited in this case. For instance, a Bayesian network is employed in the work of Kruegel et al. (2003).

While time-sliced models like dynamic Bayesian networks (Dean and Kanazawa, 1989) can capture some of these aspects, they require the specification of a time-slice width. This sampling rate must be fast enough to catch these distinctions (essentially fast enough that only one event happens between samples) and yet still long enough to make the algorithm efficient. For network traffic, with long delays between bursts of activity, this is impractical.

We seek a model that has the features described above. It should be a compact model that describes the temporal dynamics of a continuous time system. A continuous time Bayesian network is an example of this. We will give a brief introduction of CTBNs in the next chapter. After that, we will show our approaches to intrusion detection using CTBNs.

Chapter 3

Continuous Time Bayesian Networks

In this chapter, we give a brief introduction to continuous time Bayesian networks (CTBNs). A CTBN is a factorized representation of a more general dynamic process — a homogeneous Markov process (HMP). To better understand the concepts and algorithms in CTBNs, we review HMPs in Section 3.1. After that, we introduce CTBNs in Section 3.2. We then discuss phase distributions and their correlations to CTBNs in Section 3.3.

3.1 Homogeneous Markov Process

A finite-state, continuous-time Markov process is often used to model the dynamics of a system over time. It models both the temporal behavior — the expected amount of time that the system stays on the current state, and the state transitional behavior — the distribution of the target state upon transition. If the dynamics do not depend on time, such process is called a homogeneous Markov process (HMP).

3.1.1 Representation

Let X be the state variable for the system, and let n be the cardinality of its state space $\mathcal{X}$. An HMP can be compactly modeled by a start distribution $P_X(0)$ and an intensity matrix Q_X. $P_X(0)$ is a multinomial distribution over $\mathcal{X}$ indicating the probability of X starting at each of the states. Q_X models the temporal and transitional behavior of the system:

$$\mathbf{Q_X} = \begin{bmatrix} -q_{11} & q_{12} & \cdots & q_{1n} \\ q_{21} & -q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1} & q_{n2} & \cdots & -q_{nn} \end{bmatrix}.$$

q_{ii} is the intensity of X leaving state X_i. The duration of staying in the current state is exponentially distributed with parameter q_{ii}. For example, the probability density function of X staying in X_i for duration t is $P(t) = q_{ii} \exp(-q_{ii}t)$. The expected time of leaving the state X_i is $1/q_{ii}$.

q_{ij} is the intensity of X making a transition from the state X_i to X_j, when $j \neq i$. The transition follows the multinomial distribution. The probability of transition from state X_i to state X_j is $\theta_{ij} = \frac{q_{ij}}{\sum_{k \neq i} q_{ik}}$.

In an HMP, the system can only transit to the states listed in the intensity matrix, and so the intensity of leaving state X_i should be the same as all the intensities of the target states to which the system can transit from X_i. In another words, the summation of each row of the intensity matrix is always 0, or equivalently $q_{ii} = \sum_{i \neq j} q_{ij}$.

3.1.2 Query over an HMP

The distribution over the state of the process X at some future time t, $P_X(t)$, can be computed directly from $\mathbf{Q_X}$. If $P_X(0)$ is the start distribution, then

$$P_X(t) = P_X(0)\exp(\mathbf{Q_X} \cdot \mathbf{t}) \ ,$$

where $\exp$ is matrix exponential, and P_X is represented as a row vector.

It is easy to derive from the Markov property that for any time $t_1 < t_2$

$$P_X(t_2) = P_X(t_1)\exp(\mathbf{Q_X} \cdot (\mathbf{t_2} - \mathbf{t_1})) \ .$$

3.1.3 Complete Data

Complete data for an HMP are represented by a set of trajectories $\mathbf{D} = \{\tau_1, ...\tau_n\}$. Each trajectory τ_i is a complete set of state transitions: $d = \{(x_d, t_d, x'_d)\}$, meaning that X stayed in state x_d for a duration of t_d, and then transitioned to state x'_d. Therefore we know the exact state of the variable X at any time $0 \leq t \leq T$.

3.1.4 Sufficient Statistics and Likelihood

Given an HMP and its full data $\mathbf{D}$, the likelihood of a single state transition $d = \{(x_d, t_d, x'_d)\} \in \mathbf{D}$ is:

$$L_X(q, \theta : d) = (q_{x_d} \exp(-q_{x_d} t_d))(\theta_{x_d x'_d}) \ .$$

The likelihood function for $\mathbf{D}$ can be decomposed by transition:

$$\begin{aligned} L_X(q, \theta : \mathbf{D}) &= (\prod_{d \in \mathbf{D}} L_X(q : d))(\prod_{d \in \mathbf{D}} L_X(\theta : d)) \\ &= (\prod_x q_x^{M[x]} \exp(-q_x T[x]))(\prod_x \prod_{x' \neq x} \theta_{xx'}^{M[x,x']}) \ . \end{aligned}$$

If we take the log of the above function, we get the log likelihood:

$$\begin{aligned} l_X(q, \theta : \mathbf{D}) &= l_X(q : \mathbf{D}) + l_X(\theta : \mathbf{D}) \\ &= \sum_x (M[x] \ln(q_x) - q_x T[x] + \sum_{x' \neq x} M[x, x'] \ln(\theta_{xx'})) \ . \end{aligned}$$

Here $M[x, x']$ and $T[x]$ are the *sufficient statistics* of the HMP model. The sufficient statistics for a model are the summarized statistics derived from observed data that are sufficient to calculate the likelihood without having to revisit the data again. For example, for a Bernoulli distribution with parameter p, the sufficient statistics for a set of observations is just the count of number of 1's and the total number of observations. For an HMP, $M[x, x']$ is the number of times X transits from the state x to x'. We denote $M[x] = \sum_{x'} M[x, x']$. $T[x]$ is the total duration that X stays in the state x. These sufficient statistics are summaries of the trajectories in $\mathbf{D}$.

3.1.5 Learning from Complete Data

To estimate the parameters of the transition intensity matrix Q, we maximize the above log likelihood function. We have:

$$\hat{q}_x = \frac{M[x]}{T[x]}, \quad \hat{\theta} xx' = \frac{M[x, x']}{M[x]} \ .$$

3.1.6 Incomplete Data

Incomplete data in HMP are composed of a set of partially observed trajectories $\mathbf{D} = \{\tau_1^-, ... \tau_n^-\}$. Each trajectory τ_i^- consists of a set of $d = \{(S_d, t_d, dt)\}$ observations, where S_d is a *subsystem* (a nonempty subset of the states of X) of the process. Each of the triplets specifies an "interval evidence." It means that the variable X is in the subsystem S_d from time t_d to time $t_d + dt$. Some of the observations may be duration-free. *i.e.*, we only observe $X \in S_d$ at time t, but do not know how long it stayed there. This is called a "point evidence" and can be generalized using the same triplet notation described above by setting the duration to be 0. For a partially observed trajectory, we only observe sequences of subsystems, and do not observe the state transitions within the subsystems.

3.1.7 Expected Sufficient Statistics and Expected Likelihood

We can consider possible completions of a partially observed trajectory that specify the transitions that are consistent with the partial trajectory. By combining the partial trajectory and its completion, we get a full trajectory. We define $\mathbf{D}^+ = \{\tau_1^+, ..., \tau_n^+\}$ to be completions of all the partial trajectories in $\mathbf{D}$. Given a model, we have a distriubtion over $\mathbf{D}^+$, given $\mathbf{D}$.

For data $\mathbf{D}^+$, the expected sufficient statistics with respect to the probability density over possible completions of the data are $\bar{T}[x]$, $\bar{M}[x, x']$ and $\bar{M}[x]$. The expected log likelihood is

$$\begin{aligned} E[l_X(q, \theta : \mathbf{D}^+)] &= E[l_X(q : \mathbf{D}^+)] + E[l_X(\theta : \mathbf{D}^+)] \\ &= \sum_x (\bar{M}[x] \ln(q_x) - q_x \bar{T}[x] + \sum_{x' \neq x} \bar{M}[x, x'] \ln(\theta_{xx'})) \ . \end{aligned}$$

3.1.8 Learning from Incomplete Data

Expectation maximization (EM) algorithm is used to find the maximum likelihood parameters from partial trajectory. The EM algorithm iterates over the following E step and M step until the convergence on the derived likelihood function.

E step: Given the current HMP parameters, compute the expected sufficient statistics: $\bar{T}[x]$, $\bar{M}[x, x']$ and $\bar{M}[x]$ for the data set $\mathbf{D}$. This is the most complex part of the algorithm. We have a detailed description for this step alone below.

M step: From the computed expected sufficient statistics, update the new model parameters for the next EM iteration:

$$q_x = \frac{\bar{M}[x]}{\bar{T}[x]}, \quad \theta_{xx'} = \frac{\bar{M}[x, x']}{\bar{M}[x]} \ .$$

Now we show how to calculate the expected sufficient statistics using the forward-backward message passing method.

The entire trajectory $\tau \in \mathbf{D}$ can be devided into N intervals where each of the interval is separated by adjacent event changes. Assume the trajectory spans over the time interval $[0, T)$, and let $\tau[v, w]$ be the observed evidence between time v and w, including events on the time stamp v and w, and let $\tau(v, w)$ be the same set of evidence but excluding v and w. Let S be the subsystem the states are restricted on this interval.

We define

$$\alpha_{\mathbf{i}} = P(X_t = i, \tau[0, t]), \quad \beta_{\mathbf{i}} = P(\tau[t, T] \mid X_t = i)$$

Similarly, define the corresponding distribution that excludes certain point evidence as following.

$$\alpha_{\mathbf{i}}^{-} = P(X_t = i, \tau[0, t)), \quad \beta_{\mathbf{i}}^{+} = P(\tau(t, T] \mid X_t = i)$$

Denote δ_j be a vector of all 0's except for its j-th position being 1, and denote $\boldsymbol{\Delta}_{\mathbf{ij}}$ be a matrix of all 0's except that the element on i-th row and j-th column is 1.

We are now able to show the derived expected sufficient statistics.

$$\begin{aligned} E[T[x]] &= \int_0^T P(X_t \mid \tau[0, T]) \delta_x dt \\ &= \frac{1}{P(\tau[0, T])} \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} P(X_t, \tau[0, T]) \delta_x dt \end{aligned}$$

The constant fraction at the beginning of the last line serves to make the total expected time over all j sum to τ.

The integral on each interval can be further expressed as

$$\int_v^w P(X_t, \tau[0,T])\delta_x dt = \int_v^w \alpha_{\mathbf{v}} \exp(\mathbf{Q_S}(t-v))\mathbf{\Delta_{xx}} \exp(\mathbf{Q_S}(w-t))\beta_{\mathbf{w}} dt \ ,$$

where Q_S is the same as Q_X except all elements that correspond to transitions to or from S are set to 0.

$$\begin{aligned} E[M[x,x']] \quad = \quad & \frac{\mathbf{q_{x,x'}}}{P(\tau[0,T])}[\sum_{i=1}^{N-1} \alpha_{\mathbf{t_i}}^{-} \mathbf{\Delta_{x,x'}} \beta_{\mathbf{t+i}}^{+} + \\ & \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} \alpha_{\mathbf{t_i}} \exp(\mathbf{Q_S}(t-t_i))\mathbf{\Delta_{x,x'}} \exp(\mathbf{Q_S}(t_{i+1}-t))\beta_{\mathbf{t_{i+1}}} dt] \end{aligned}$$

The integrals appearing in $E[T]$ and $E[M]$ can be computed via the standard ODE solver, like the Runge-Kutta method (Press et al., 1992).

Now the only remaining problem is to calculate α and β. Let $\mathbf{Q_{SS'}}$ be the transitioning intensity matrix of the HMP from one subsystem S to another S'. This matrix is the same as Q_X, but only elements corresponding to transitions from S to S' are non-zero.

$$\alpha_{\mathbf{t_{i+1}}} = \alpha_{\mathbf{t_i}} \exp(\mathbf{Q_{S_i}}(t_{i+1}-t_i))\mathbf{Q_{S_i S_{i+1}}} \ ,$$

$$\beta_{\mathbf{t_i}} = \mathbf{Q_{S_{i-1} S_i}} \exp(\mathbf{Q_{S_i}}(t_{i+1}-t_i))\beta_{\mathbf{t_{i+1}}} \ .$$

During this forward-backward calculation, it is also trivial to answer queries such as

$$P(X_t = x \mid \tau[0,T]) = \frac{1}{P(\tau)} \alpha_{\mathbf{t}}^{-} \mathbf{\Delta_{xx}} \beta_{\mathbf{t}} \ .$$

3.1.9 Limitation

While HMPs are good for modeling many dynamic systems, they have their limitations when the systems have multiple components and the state space grows exponentially as the number of variables increases. An HMP does not model the variable independencies and therefore it has to use a unified state X to represent the joint behavior of all the involving components in the system. In the next section, we show how a continuous time Bayesian network can be used to address this issue.

3.2 Continuous Time Bayesian Networks

Nodelman et al. (2002) extend the theory of HMPs and present continuous time Bayesian networks (CTBNs), which model the joint dynamics of several local variables by allowing the transition model of each local variable X to be a Markov process whose parametrization depends on some subset of other variables U.

3.2.1 Definition

We first give an definition of an inhomogeneous Markov process called a conditional Markov process. It is a critical concept for us to formally introduce the CTBN framework.

Definition 1 *Nodelman et al. (2003)* A conditional Markov process X *is an inhomogeneous Markov process whose intensity matrix varies as a function of the current values of a set of discrete conditioning variables* U*. It is parametrized using a conditional intensity matrix*

(CIM) $Q_{X|U}$ – a set of homogeneous intensity matrices $Q_{X|u}$, one for each instantiation of values u to U.

We call U, the *parents* of X. When the set of U is empty, the CIM is simply a standard intensity matrix.

CIMs provide a way to model the temporal behavior of one variable conditioned on some other variables. By putting these local models together, we have a joint structured model — a continuous time Bayesian network.

Definition 2 *Nodelman et al. (2003)* A continuous time Bayesian network $\mathcal{N}$ *over* $\boldsymbol{X}$ *consists of two components: an initial distribution $P^0_{\boldsymbol{X}}$, specified as a Bayesian network $\mathcal{B}$ over a set of random variables $\boldsymbol{X}$, and a continuous transition model, specified using a directed (possibly cyclic) graph $\mathcal{G}$ whose nodes are $X \in \boldsymbol{X}$; $\boldsymbol{U}_X$ denotes the parents of X in $\mathcal{G}$. Each variable $X \in \boldsymbol{X}$ is associated with a conditional intensity matrix, $\boldsymbol{Q}_{X|\boldsymbol{U}_X}$.*

The dynamics of a CTBN are quantitatively defined by a graph. The instantaneous evolution of a variable depends only on the current value of its parents in the graph. The quantitative description of a variable's dynamics is given by a set of intensity matrices, one for each value of its parents. That means the transition behavior of the variable is controlled by the current values of its parents. If we amalgamate all the variables in the CTBN together, we get a single homogeneous Markov process over the joint state space.

3.2.2 Learning

In the context of CTBNs, the model parameters consist of the CTBN structure $\mathcal{G}$, the initial distribution P_0 parameterized by a regular Bayesian network, and the conditional intensity matrices (CIMs) of each variable in the network. In this section, we assume the CTBN structure is known to us, so we only focus on the parameter learning. We also assume the model is irreducible. So the initial distribution P_0 becomes less important in the context of CTBN inference and learning, especially when the time range becomes significantly large. Therefore, parameter learning in our context is to estimate the conditional intensity matrices $Q_{X_i|\mathbf{U_i}}$ for each variable X_i, where $\mathbf{U_i}$ is the set of parent variables of X_i.

Learning from Complete Data

Complete data in a CTBN are represented by a set of trajectories $\mathbf{D} = \{\tau_1, ...\tau_n\}$. Each trajectory τ_i is a complete set of state transitions and the times at which they occurred. Therefore we know full instantiations to all the variable at any time.

Nodelman et al. (2003) presented an efficient way to learn a CTBN model from fully observed trajectories.

With complete data, we know full instantiations to all the variables for the whole trajectory. So we know which CIM is governing the transition dynamics of each variable at any time. The sufficient statistics are $M[x, x'|u]$ — the number of times X transits from the state x to x' given its parent instantiation u — and $T[x|u]$ — the total duration that X stays in the state x given its parent instantiation u. We denote $M[x|u] = \sum_{x'} M[x, x'|u]$.

The likelihood function for $\mathbf{D}$ can be decomposed to

$$L_{\mathcal{N}}(q, \theta : \mathbf{D}) = \prod_{X_i \in \mathbf{X}} L_{X_i}(q_{X_i|U_i} : \mathbf{D}) L_{X_i}(\theta_{X_i|U_i} : \mathbf{D}))\ ,$$

Where

$$L_X(q_{X|\mathbf{U}} : \mathbf{D}) = \prod_{\mathbf{u}} \prod_{x} q_{x|\mathbf{u}}^{M[x|\mathbf{u}]} \exp(-q_{x|\mathbf{u}} T[x|\mathbf{u}])\ ,$$

and

$$L_X(\theta : \mathbf{D}) = \prod_{\mathbf{u}} \prod_{x} \prod_{x' \neq x} \theta^{M}_{xx'|\mathbf{u}}[x, x'|\mathbf{u}]\ .$$

If we put the above functions together and take the log, we get the log likelihood:

$$\begin{aligned} l_X(q, \theta : \mathbf{D}) &= l_X(q : \mathbf{D}) + l_X(\theta : \mathbf{D}) \\ &= [\sum_{\mathbf{u}} \sum_{x} M[x|u] \ln(q_x|u) - q_{[x|u]} T[x|u]] \\ &\quad +[\sum_{\mathbf{u}} \sum_{x} \sum_{x' \neq x} M[x, x'|u] \ln(\theta_{xx'|u}))]. \end{aligned} \tag{3.1}$$

By maximizing the above log likelihood function, the model parameters can be estimated as

$$\hat{q}_{x|\mathbf{u}} = \frac{M[x|\mathbf{u}]}{T[x|\mathbf{u}]}, \quad \hat{\theta}_{xx'|\mathbf{u}} = \frac{M[x, x'|\mathbf{u}]}{M[x|\mathbf{u}]}\ .$$

Learning from Incomplete Data

Nodelman et al. (2005b) present the expectation maximization (EM) algorithm to learn a CTBN model from partially observed trajectories $\mathbf{D}$.

The expected sufficient statistics are $\bar{M}[x, x'|\mathbf{u}]$, the expected number of times that X transits from state x to x' when its parent set $\mathbf{U}$ takes the values $\mathbf{u}$, and $\bar{T}[x|\mathbf{u}]$, the expected amount of time that X stays in the state x under the same parent instantiation. We denote $\bar{M}[x|\mathbf{u}]$ to be $\sum_{x'} \bar{M}[x, x'|u]$. The expected log likelihood can be decomposed in the same way as in Equation (3.1), except that the sufficient statistics $M[x, x'|u]$, $T[x|u]$ and $M[x|u]$ are now replaced with expected sufficient statistics $\bar{M}[x, x'|u]$, $\bar{T}[x|u]$ and $\bar{M}[x|u]$.

The EM algorithm for a CTBN works essentially in the same way as for an HMP. The expectation step is to calculate the expected sufficient statistics using inference method (will be described in Section 3.2.3). The maximization step is to update the model parameters like this:

$$\hat{q}_{x|\mathbf{u}} = \frac{\bar{M}[x|\mathbf{u}]}{\bar{T}[x|\mathbf{u}]}, \quad \hat{\theta}_{xx'|\mathbf{u}} = \frac{\bar{M}[x, x'|\mathbf{u}]}{\bar{M}[x|\mathbf{u}]} \; .$$

3.2.3 Inference

Now given a CTBN model and some observed data, we would like to query the model. This task is usually called an inference problem. For example, calculating the expected sufficient statistics given a model is an inference task.

Exact inference

Nodelman et al. (2005b) provide an exact inference algorithm using expectation maximization to reason and learn the parameters from partially observed data. This exact inference algorithm requires flattening all the variables into a single Markov process, and perform inference as in an HMP. It has the problem that it makes the state space grow exponentially large. Therefore, the exact inference method is only feasible for problems with very small state spaces.

Approximate inference

Because of the issue addressed below, much work has been done on CTBN approximate inference. Nodelman et al. (2005a) present an expectation propagation algorithm. Saria et al. (2007) give another message passing algorithm that adapts the time granularity. Cohn et al. (2009) provide a mean field variational approach. El-Hay et al. (2008) show a Gibbs sampling method approach using Monte Carlo expectation maximization. Fan and Shelton (2008) give another sampling based approach that uses importance sampling.

To estimate the parameters of the models we build for the two applications (NIDS and HIDS), we employ inference algorithms including exact inference and a Rao-Blackwellized particle filtering (RBPF) algorithm, depending on the model size. RBPFs have been widely employed mainly in the area of vision and robot mapping. For example, Khan et al. (2004) track a target in a clutter and Schulz et al. (2003) track people's locations using a network of sensors. Sim et al. (2007) summarize different models of RBPF for vision-based SLAM.

Doucet et al. (2000) propose the RBPF algorithm for dynamic Bayesian networks that work in discrete time fashion by exploiting the structure of the DBN. They apply their method on the problem of online regression and robot localization. Ng et al. (2005) extend RBPF to continuous time dynamic systems and apply the method to the K-9 experimental Mars rover at NASA Ames Research Center. Their model is a hybrid system containing both discrete and continuous variable. They use particle filters for the discrete variables and unscented filters for the continuous variable. Our work are similar to this work in the method of applying RBPF to CTBNs, but our model contains only discrete variables and our evidence is over continuous intervals.

3.3 Phase distributions

For a normal CTBN, the duration of variable X staying at state x given its parent instantiation u is a simple exponential distribution with parameter $q_{x|u}$. But in some applications the duration is not exponentially distributed. Therefore, we need to use distributions that have better expressive power than an exponential distribution. Phase distributions provide us a natural way to model more complex durations.

3.3.1 Definition

The class of phase distributions is a highly flexible family. It models a (continuous time) Markov process which evolves through a set of phases, terminating in an absorption state.

Each of these phases has an associated exponential distribution, which describes the duration of time that the process stays in that phase. The model as a whole can be described as a (possibly cyclic) graph of phases. The process transits through this graph, going through some or all of these phases and finally leaves (to an absorbing state). The distribution of when the process leaves this system is called a *phase distribution*.

Definition 3 *A* phase distribution *of p phases is defined as the distribution over the time when a homogeneous Markov process with a single absorbing state and p transient phases reaches absorption (Neuts, 1975, 1981).*

With a finite number of phases, any distribution with support contained entirely in the positive real number line or something similar can be approximated with arbitrary precision by a phase distribution (Neuts, 1975).

3.3.2 CTBN Durations as Phase Distributions

Nodelman et al. (2005b) show how to model the duration of a CTBN variable X as a phase distribution by allowing a single state to have multiple phases. The phase distribution parameters of X depend on the state of its parent instantiation but not the phase of the parent variables. If the parent instantiation changes, one might allow X in its current state to stay in the same phase or to reset. A normal phase distribution has an absorption state which indicates that the process leaves the system. But in this context, no absorption state is used. The leaving intensity simply means that X has transited to a new state.

Instead of using a phase distribution to model the complex duration of $X = x|u$, we can also introduce a hidden variable H_x as X's new parent. H_x does not depend on X's original parents. We can amalgamate all the intensities of H_x and X into a single cluster node S, and then use phase distribution to model X, with $|Val(H_x)|$ phases per state. However, phase distributions have more expressive power than using hidden variables in this case. By using a hidden variable H_x, the amalgamated intensity matrix for S has to have 0's for transitions where more than one variable in S are changing. However, a general phase distribution does not have this constraint.

3.3.3 Parameter learning

Now we introduce how to learn paramters for a phase distribution in general.

A phase distribution has three parameters:

- Number of phases: $p, \;\; 0 < p < \infty$
- Initial distribution: π, a p-dimensional column vector
- Transition model: Q, a p-by-p intensity matrix

We will assume p to be fixed (or given). We will learn π and Q from data. The interpretation of π is relatively straight-forward. It is a distribution over the starting state for the Markov chain. Its elements are all non-negative and they sum to 1. Q is more complicated and has a few restrictions:

- $Q_{ii} \leq 0$: Its absolute value is the intensity of leaving phase i. $1/|Q_{ii}|$ is the mean time spent in state i.

- $Q_{ij} \geq 0$: Its value is the intensity from phase i to phase j, for $i \neq j$. The ratio of Q_{ij} to $|Q_{ii}|$ is equal to the probability that state j will immediately follow state i.

- $|Q_{ii}| \geq \sum_{j \neq i} Q_{ij}$: The difference between the two sides of the inequality is the intensity of a transition out of the system to the absorbing state.

The cumulative distribution and probability density of a phase distribution can respectively be written as

$$F(\tau) = 1 - \pi^\top e^{Q\tau}\mathbf{1}, \;\; \tau \geq 0$$

$$f(\tau) = -\pi^\top e^{Q\tau} Q\mathbf{1}, \;\; \tau \geq 0 \; .$$

The exponentiation is the matrix exponential and the multiplications are matrix-vector multiplications. $\mathbf{1}$ is the vector of all 1s.

We can employ the expectation-maximization (EM) algorithm (Dempster et al., 1977) to find the parameters that maximize the likelihood of the data. This method first appeared in Asmussen et al. (1996). We duplicate the final results here for clarity and completeness.

Each data point, τ_k, is the total time from starting the process until entering the absorbing state. The hidden information is the sequence of transient states and the amount of time spent in each one. For a phase distribution, the sufficient statistics are

- T_i: the amount of time spent in state i,

- M_{ij}: the number of times the system transitioned from state i to state j,

- M_i: the number of times the system transitioned out of state i, and

- N_i: the number of times the system started in state i.

For this use of EM, the steps are as follows:

E-step: Given the phase distribution parameters π and Q, and the dataset $D = \{\tau_1, \tau_2, \ldots, \tau_S\}$, calculate the expected sufficient statistics: $\bar{E}[T_i|D]$, $\bar{E}[M_{ij}|D]$, $\bar{E}[M_i|D]$, $\bar{E}[N_i|D]$.

M-step: Using the calculated expected sufficient statistics, update the parameters π and Q to increase the likelihood of the data.

For the *E-step*, the sufficient statistics calculations can be performed as follows. We let $\boldsymbol{r} = -Q\mathbf{1}$ and δ_i be a vector of all zeros except a single 1 in location i.

$$\begin{aligned}
\bar{E}[T_i|D] &= \sum_{k=1}^{S} \frac{\int_0^{\tau_k} \pi^\top e^{Qt} \delta_i \delta_i^\top e^{Q(\tau_k - t)} \boldsymbol{r} dt}{\pi^\top e^{Q\tau_k} \boldsymbol{r}} \\
\bar{E}[M_{ij}|D] &= \sum_{k=1}^{S} \frac{\int_0^{\tau_k} \pi^\top e^{Qt} \delta_i Q_{ij} \delta_j^\top e^{Q(\tau_k - t)} \boldsymbol{r} dt}{\pi^\top e^{Q\tau_k} \boldsymbol{r}} \\
\bar{E}[M_i|D] &= \sum_{j \neq i} M_{ij} + \sum_{k=1}^{S} \frac{\pi^\top e^{Q\tau_k} \delta_i r_i}{\pi^\top e^{Q\tau_k} \boldsymbol{r}} \\
\bar{E}[N_i|D] &= \sum_{k=1}^{S} \frac{\pi_i \delta_i^\top e^{Q\tau_k} \boldsymbol{r}}{\pi^\top e^{Q\tau_k} \boldsymbol{r}}
\end{aligned}$$

The integrals in these formulas can be computed via the Runge-Kutta method like those for a general HMP (Press et al., 1992).

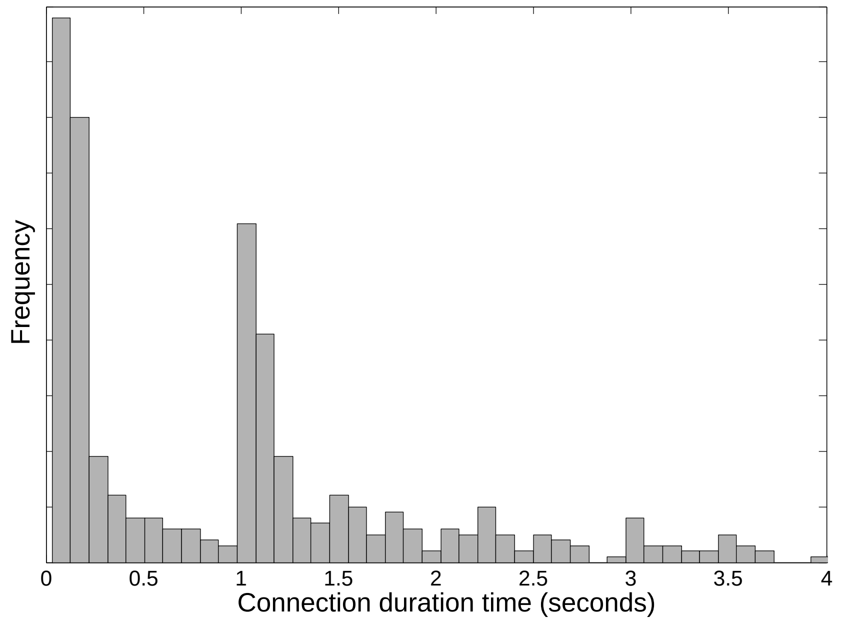


Figure 3.1: Example histogram of TCP connection durations for normal network traffic.

For the *M-step*, the maximizations are as follows.

$$\pi_i = \frac{\bar{E}[N_i|D]}{S}$$
$$Q_i = -\frac{\bar{E}[M_i|D]}{\bar{E}[T_i|D]}$$
$$Q_{ij} = \frac{\bar{E}[M_{ij}|D]}{\bar{E}[T_i|D]}, \;\; i \neq j$$

3.3.4 Fitting Phase Distributions to Network Features

We discover that much work studying network traffic behavior assumes the connection duration times distributed exponentially. This is not always true. Figure 3.1 shows a histogram of the connection times for normal network traffic from "The Forbidden City" dataset. Courtesy of Intel. This distribution is multi-modal. Its non-exponential character attests to the inherent

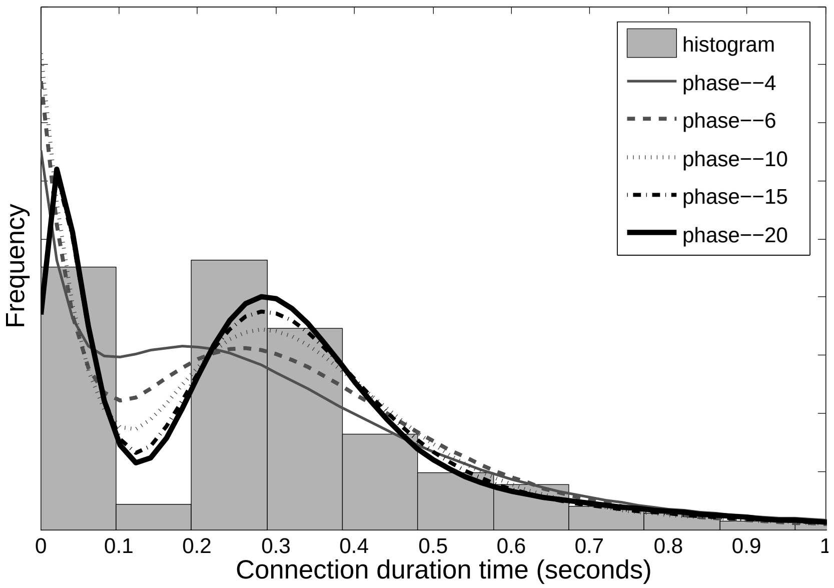


Figure 3.2: A comparison of the fit of phase distributions, varying the number of phases, p.

state within a TCP connection. The protocol itself has state, and obviously the communicating computers have state of their own. This inspires us to use a more complex duration model if we want to model the network features like connection times, for example, a phase distribution.

EM for phase distributions is sensitive to the parameter initialization. The number of parameters grows quadratically with the number of phases. Selecting a suitable value for p is important. In Figure 3.2 we can see that an increase in the number of phases results in a better fit. However, we must trade this off against an increase in the learning time and amount of data required.

We follow the lead of Asmussen et al. (1996) and initialize the transition matrix, Q, to be a *Coxian* distribution, a subclass of phase distributions, in which the graph of state transitions forms a chain (with transitions to the absorption state possible from each phase).

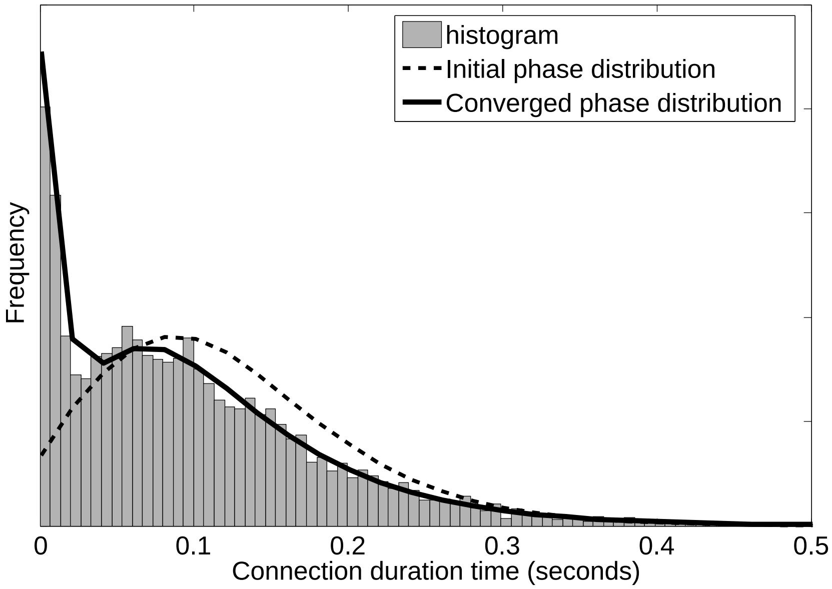


Figure 3.3: Convergence for fitting phase distributions via EM.

We initialize the expected holding time for each state to be $E[\tau]/p$. In order to allow the learning to explore the full space of phase distributions, we add noise to elements of Q, thus resulting in a general phase distribution. Our experiments shows that initializing Q as an approximate Coxian distribution leads us faster convergence. Figure 3.3 demonstrates the effects of the EM procedure by illustrating the model fit after initialization and then after the full EM algorithm.

Other models have also been employed on network traffic data. The *Erlang* distribution is widely used in the field of stochastic processes. Erlang distributions are a subclass of phase distributions in which the graph of state transitions forms a chain and no state, except the last one, can lead to the absorbing state. The probability density function of an Erlang distribution with parameters k and λ is $f(x; k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}$, for $x > 0$. When $k = 1$, the Erlang distribution simplifies to an exponential distribution. The *Levy* distribution, which

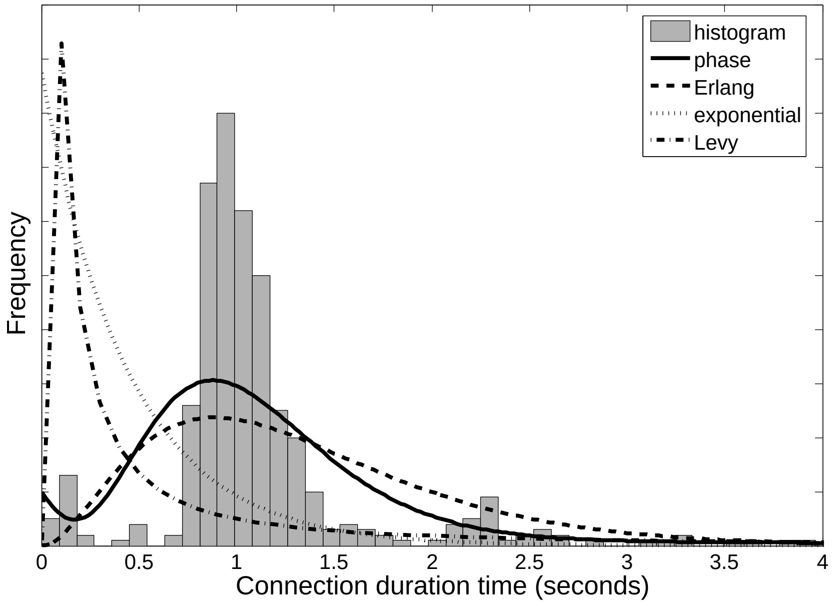


Figure 3.4: Empirical distribution and model fit for host1.

is one of the few stable distributions, is skewed and heavy tailed. It has also been used for describing the underlying distribution of network traffic duration time. Its density function is $f(x;c) = \sqrt{\frac{c}{2\pi}} \; \frac{e^{-c/2x}}{x^{3/2}}$ with parameter c.

We fit the Erlang, exponential, and Levy distributions separately on our real data and compare their fitting results with phase distribution. Figures 3.4 and 3.5 demonstrate the flexibility of phase distribution. The two figures are for different hosts. Note that the phase distribution fits either dataset well, while no other distribution works well for both. The Levy distribution is a particularly poor fit because it is heavy-tailed and thus the parameter c is chosen based mainly on the tails of the empirical distribution.

We also try to fit a phase distribution on other network features, for example, the number of connections within a time window (Agosta et al., 2007a). Figure 3.6 shows that the fit is also good. All these experiments not only demonstrate the expressive ability of phase distri-

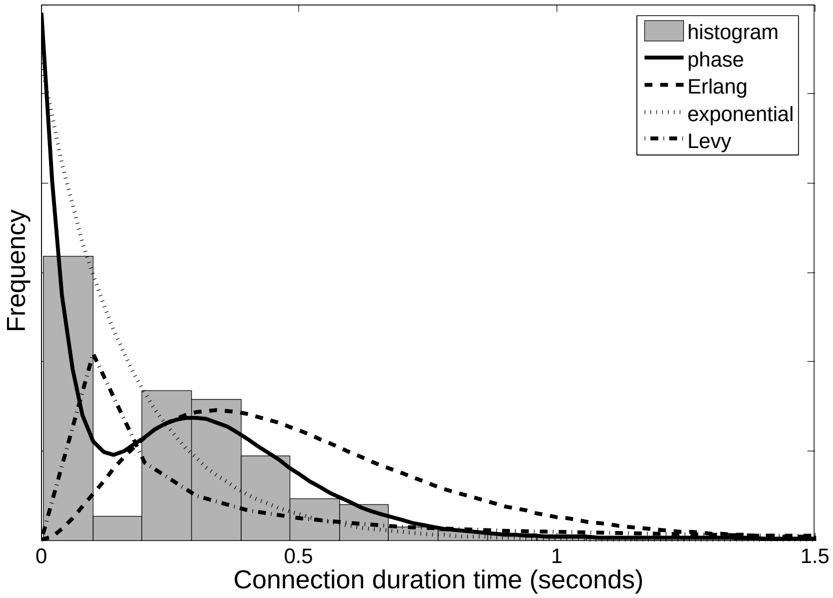


Figure 3.5: Empirical distribution and model fit for host2.

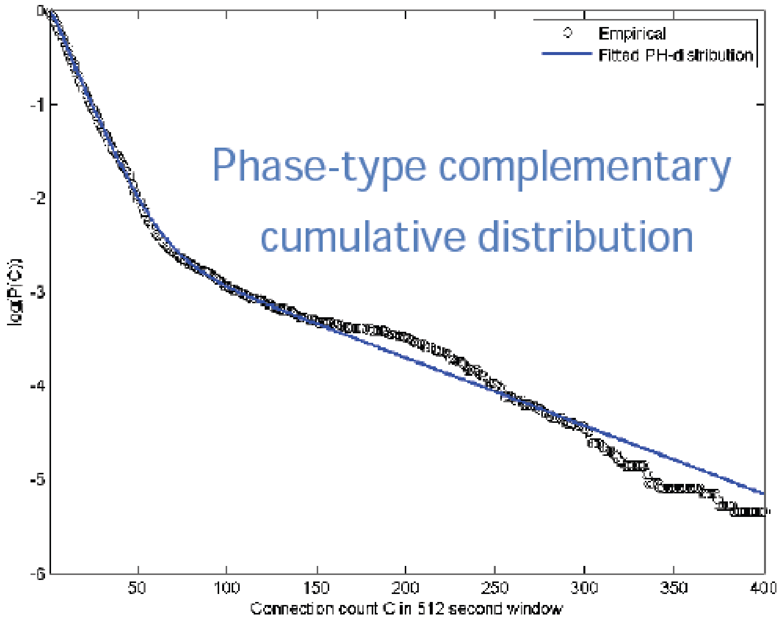


Figure 3.6: Fitting a phase distribution to connection counts within a window

butions, but also motivate us to use complex duration models other than a simple exponential distribution to describe computer activity features.

As mentioned earlier in Section 3.3.2, using a phase distribution to model the durations of a variable is equivalent to introducing a hidden variable as a parent. Therefore, we can also build a CTBN model containing the computer activity, i.e. connection times, as a variable, and add a hidden variable as its parent. In this way, the duration of the variable is conditioned on its parent instantiation and is no longer marginally exponentially distributed. We will demonstrate how we build such a CTBN model in the following chapter.

Chapter 4

NIDS - Basic Model and Inference

In this chapter, we describe our approach for network intrusion detection using network data (NIDS). We employ CTBNs to model the normal TCP packet header traces of a single host, and flag possible intrusions based on the deviation of its future traces from this norm. We build two detectors that employ different CTBN models: a factored one and a connected one. In Section 4.1, we present the factored model and how we learn its parameters. In Section 4.2, we present the connected model and its parameter estimation. In Section 4.3 we describe how to detect intrusions based on the model we build.

4.1 A Factored CTBN Model

Our first successful detector uses a factored CTBN model to describe the network traffic (Xu and Shelton, 2008).

PORT	DESCRIPTION
80	World Wide Wed HTTP
139	NETBIOS Session Service
443	HTTP protocol over TLS/SSL
445	Microsoft-DS
1863	MSNP
2678	Gadget Gate 2 Way
1170	AT+C License Manager
110	Post Office Protocol - Version 3

Figure 4.1: Ranking of the most frequent ports on LBNL dataset

PORT	DESCRIPTION
80	World Wide Web HTTP
8080	HTTP Alternate
443	HTTP protocol over TLS/SSL
113	Authentication Service
5101	Talarian TCP
995	pop3 protocol over TLS/SSL
51730	*unknown*
59822	*unknown*

Figure 4.2: Ranking of the most frequent ports on WIDE dataset.

4.1.1 CTBN Model for Network Traffic

As described in Section 3.2, CTBNs are well suited for describing such structured stochastic processes with finitely many states that evolve over continuous time. By using a CTBN to model the network traffic activities, we can capture the complex context of network behaviors in a meaningful and hierarchical way.

A typical machine in the network may have diverse activities with various service types (*e.g.* HTTP, mail server). Destination port numbers describe the type of service a particular network activity belongs to. Some worms usually propagate malicious traffic towards certain well known ports to affect the quality of the services who own the contact ports. By looking

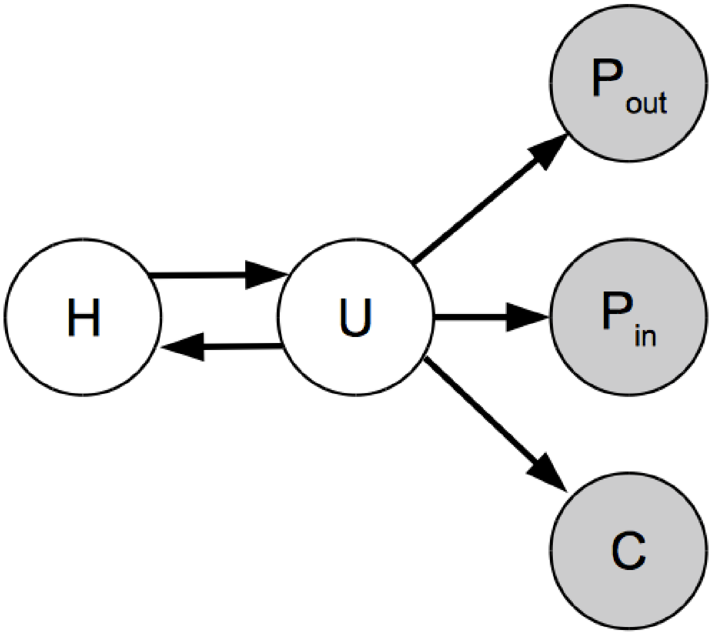


Figure 4.3: CTBN port level submodel; the whole model contains 9 of such submodels.

at traffic associated with different ports we are more sensitive to subtle variations that do not appear if we aggregate trace information across ports. Figure 4.1 and Figure 4.2 show the most popular ports ranked by their frequencies in the network traffic on the datasets we use (described in more depth later). These services are, to some extent, independent of each other. We therefore model each port's traffic with its own CTBN submodel.

Inside our port-level submodel, we have the fully observed node C — the number of concurrent connections active on the host — and nodes packet-in P_i and packet-out P_o — the transmission of a packet to or from the host. P_i and P_o have no intrinsic state: the transmission of a packet is an essentially instantaneous event. Therefore they have events (or "transitions") without having state. We discuss this further in the next subsection.

To allow complex duration distributions for the states of variables, we introduce another two nodes H and U. U is a partially observed variable which we call the mode whose state indicates whether the model can next send a packet, receive a packet, start a new connection, or terminate a connection. It therefore has four states, and we limit the conditional rates of the observed variables (C, P_o, and P_i) to be zero if the current mode differs from the variable's activity type. Therefore, U is observed when an event in C, P_o and or P_i occurs, but is unobserved between events.

H is a hidden variable that models the internal state and intrinsic features of the host machine. For the experiments we show later, we let H be a binary process. While the arrival times for events across an entire CTBN are distributed exponentially (as the entire system is Markovian), once certain variables are unobserved (like H), the marginal distribution over the remaining variables is no longer Markovian. These nodes make up the structure shown in Figure 4.3. We feel this model represents a good balance between descriptive power and tractability. If the model contains more variables, exact inference will become intractable because its state space grows exponentionally with the number of nodes.

4.1.2 Adding Toggle Variables to CTBNs

As mentioned above, the variables P_o and P_i do not have state, but rather only events. To describe such transitionless events, we set P_o and P_i to be "toggle variables." That is, they have two indistinguishable states. As a binary variable, they have two parameters for each parent instantiation (the rate of leaving state 0 and the rate of leaving state 1) and we require these two parameters to be the same. A packet event, therefore, consists of flipping the state

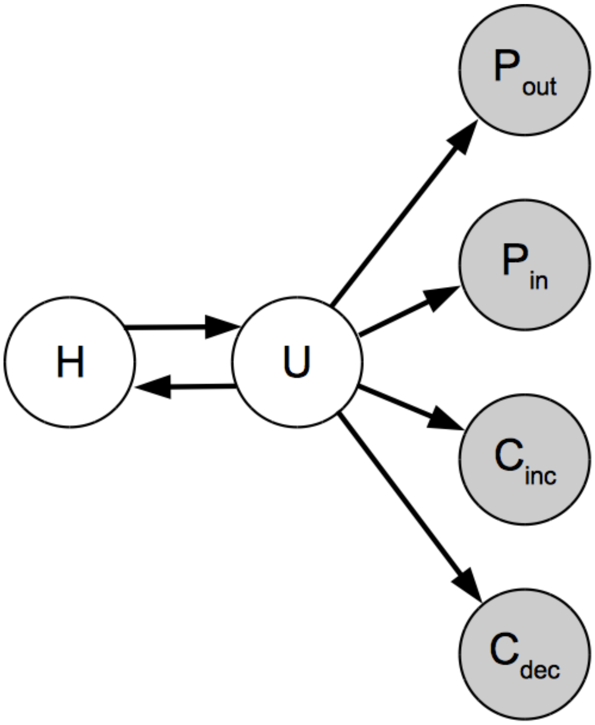


Figure 4.4: The equivalent CTBN port level submodel

of the corresponding packet variable.

The concurrent connection count variable, C, also poses a slight modeling problem for CTBNs. As originally presented, CTBNs can only deal with finite-domain variables. Although most of the operating systems do have a limit on the number of concurrent connections, this number can potentially be extremely large. Our traffic examples do exhibit a wide range of concurrent connection counts. We tried quantizing C into fixed bins, but the results were fairly poor. We instead note that C can only increase or decrease by one at any given event (the beginning or ending time of a connection). Furthermore, we make the assumption that the arrival of a new connection and the termination of an existing connection are both independent of the number of other connections. This implies that the intensity with which *some* connection starts (stops) is same as any other connections. C is thus a random walk constrained to the non-negative integers.

Let Q_{inc} be the intensity for the arrival of a new connection, and Q_{dec} be the intensity for the termination of a new connection. Let $Q_{change} = Q_{dec} + Q_{inc}$. The resulting intensity matrix has the form:

$$\mathbf{Q_{C|m}} = \begin{pmatrix} & \ddots & & & & & \\ 0 & Q_{dec} & -Q_{change} & Q_{inc} & 0 & \cdots & \\ \cdots & 0 & Q_{dec} & -Q_{change} & Q_{inc} & 0 & \cdots \\ & \cdots & 0 & Q_{dec} & -Q_{change} & Q_{inc} & 0 \\ & & & & & \ddots & \end{pmatrix}.$$

Note that the only free parameters in the above matrix are Q_{inc} and Q_{dec}. Therefore, this model is the same as one in which we replace C with two toggle variables C_{inc} and C_{dec}. C_{inc} and C_{dec} operate like P_o and P_i above: their exact state does not matter, it is the transition between states that indicates an event of interest. The model structure is shown in Figure 4.4.

4.1.3 An Extended CTBN Model

CTBN models assume that no two variables change at exactly the same instant. However, we might like U and H to change at the same time. As they both represent abstract concepts, there is no physical reason why they should not change simultaneously (they represent different abstract attributes about the machine's internal state).

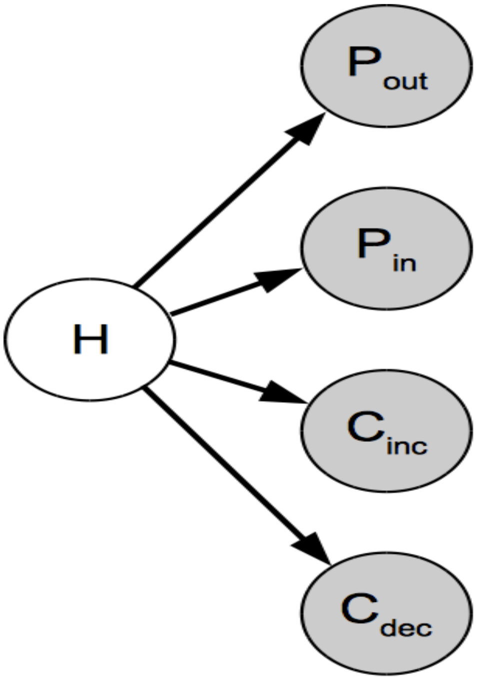


Figure 4.5: The extended CTBN port level submodel

The model in Figure 4.4 has 36 independent parameters.[1] Allowing U and H to change simultaneously requires introducing 24 new parameters: for each of the 8 states for U and H there are 3 new transition possibilities.

Equivalently, we can use the CTBN model shown in Figure 4.5 where H now has 8 states. However, this diagram does not demonstrate all of the structure. The toggle variables (P_o, P_i, C_{inc}, and C_{dec}) are each allowed to change only for 2 of the states of H (the two that corresponded to U from the previous model having the correct mode) and they are required to have the same rate for both of these states.

We have also considered other extensions to the model. For instance, it might be natural

[1] Each toggle has only 1 because it can only change for one setting of U. An intensity matrix for U has 12 independent parameters for each of the two values of H and similarly H has 2 independent parameters for each of the four values of U.

to allow the packet rate to have a linear dependence on the number of connections. However, in practice, this extension produced worse results. It seems that the rate is more of a function of the network bandwidth and the currently transmitting application that generates packets than the number of concurrent connections on a port.

4.1.4 Parameter Estimation

Our partially observed trajectory (H is unobserved, U is partially observed) specifies a sequence of subsystems of this process, each with an associated duration. We use the expectation maximization (EM) algorithm to estimate the parameters (see Chapter 3 for more details).

The ESS for any variable X in a CTBN are $\bar{T}_{X|\mathbf{U}}[x|\mathbf{u}]$, the expected amount of time X stays at state x given its parent instantiation $\mathbf{u}$, and $\bar{M}_{X|\mathbf{U}}[x, x'|\mathbf{u}]$, the expected number of transitions from state x to x' given X's parent instantiation $\mathbf{u}$.

For our new types of variables (toggle variables) P_i, P_o, C_{inc} and C_{dec}, we need to derive different sufficient statistics. They are quite similar so we just take P_i as an example. Let $\mathbf{U_{P_i}}$ be its parent U's instantiation when event P_i can happen. The ESS we need are $\bar{M}_{P_i|\mathbf{U_{P_i}}}$: the expected number of times P_i changes conditioned on it parent instantiation $\mathbf{U_{P_i}}$. Since event P_i can only occur when $\mathbf{U} = \mathbf{U_{P_i}}$, the ESS $\bar{M}_{P_i|\mathbf{U_{P_i}}}$ is just M_{P_i}, the total number of times P_i changes. We also need the total expected amount of time that packet-in occurred while $\mathbf{U} = \mathbf{U_{P_i}}$.

In EM, we use the ESS as if they were the true sufficient statistics to maximize the likelihood with respect to the parameters. For a “regular” CTBN variable X (such as our hidden variable H and M), the following equation performs the maximization.

$$Q_X|u(x, x') = \frac{\bar{M}[x, x'|u]}{\bar{T}[x|u]}$$

For our new toggle variable, i.e. P_i, the maximization is

$$Q_{P_i|\mathbf{u}} = \frac{M_{P_i}}{T[U = \mathbf{u}]}$$

The above sufficient statistics can be calculated using the exact inference algorithm of Nodelman et al. (2005b).

4.2 A Connected CTBN Model

A second successful detector we build uses a connected CTBN model.

4.2.1 CTBN Model for Network Traffic

We use the same port-level submodel as our extended factored model (see Section 4.1.3). We have a latent variable H, which has 8 states that represent different abstract attributes about the machine's internal state, and four fully observed toggle variables: $P_{in}, P_{out}, C_{inc}, C_{dec}$, which are each allowed to change only for 2 of the states of H and required to have the same rate for both of these states.

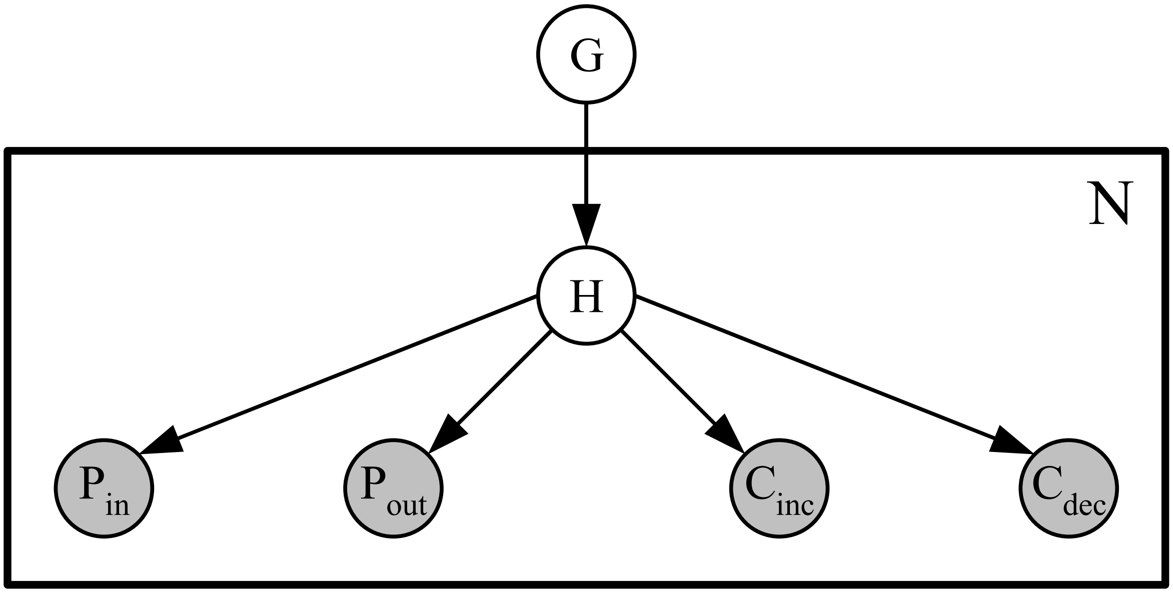


Figure 4.6: CTBN model for network traffic as a plate model. N is the number of port.

In the factored model described earlier, we assumed that the traffic associated with different ports are independent of each other, so the port-level submodels are isolated. Here we remove this restriction by introducing another latent variable G that ties the port submodels together. Figure 4.6 shows the full model as a duplication of N such plate models.

4.2.2 Parameter Learning using RBPF

To calculate the expected sufficient statistics in the E-step of EM for parameter learning, the exact inference algorithm in Nodelman et al. (2002) flattens all the variables into a joint intensity matrix and reasons about the resulting homogeneous Markov process. The time complexity is exponential in the number of variables. For example, if there are 9 port models, the network contains 46 variables in total.

We notice that our model has a nice tree structure which makes Rao-Blackwellized particle filtering (RBPF) a perfect fit. RBPF uses a particle filter to sample a portion of the variables and analytically integrates out the rest. It decomposes the model structure efficiently and thus reduces the sampling space.

We denote τ as the whole observed traffic sequences on the particular host, and τ_j as the traffic associated with port j. If we denote N port-level hidden variables as $H_1, ..., H_N$, the posterior distribution of the whole model can be factorized as $P(G, H_1, ..., H_N \mid \tau) = P(G \mid \tau) \prod_{i=1}^{N} P(H_i \mid G, \tau)$. We use a particle filter to estimate G's conditional distribution $P(G \mid \tau)$ as a set of sampled trajectories of G. It is difficult to sample directly from the posterior distribution. We use an importance sampler to sample a particle from a proposal distribution and the particles are weighted by the ratio of its likelihood under the posterior distribution to the likelihood under the proposal distribution (Fan and Shelton, 2008). Since the variable G is latent and has no parents, we can use forward sampling to sample the particles from $P(G)$. Each port-level submodel is then independent from the rest of the network, given full trajectory of G.

The expected sufficient statistics (ESS) for any variable X in a CTBN are $\bar{T}_{X|\mathbf{U}}[x|\mathbf{u}]$, the expected amount of time X stays at state x given its parent instantiation $\mathbf{u}$, and $\bar{M}_{X|\mathbf{U}}[x, x'|\mathbf{u}]$, the expected number of transitions from state x to x' given X's parent instantiation $\mathbf{u}$. Let $g^i \sim P(G)$, $i = 1, \ldots, M$ be the particles. We define their likelihood weights to be $w_i = \frac{P(g^i|\tau)}{P(g^i)}$ and let $W = \sum_i w_i$ be the sum of the weights. Then general importance sampling allows that an expected sufficient statistic can be estimated in the following way, where SS

is any sufficient statistic.

$$
\begin{aligned}
&E_{(g,h_1,\ldots,h_N)\sim P(G,H_1,\ldots,H_N|\tau)}[SS(g,h_1,\ldots,h_N)] \\
&\qquad = E_{g\sim P(G|\tau)}E_{h_1,\ldots,h_N\sim P(H_1,\ldots,H_N|g,\tau)}[SS(g,h_1,\ldots,h_N)] \\
&\qquad \approx \frac{1}{W}\sum_i w_i E_{h_1,\ldots,h_N\sim P(H_1,\ldots,H_N|g^i,\tau)}[SS(g^i,h_1,\ldots,h_N)]
\end{aligned}
$$

The expected sufficient statistics of the whole model are in two categories: those that depend only on g, $ESS(g)$, and those that depend on a port model k, $ESS(g,h_k,\tau_k)$. $ESS(g)$ is simply the summation of counts (the amount of time G stays at some state, or the number of times G transits from one state to another) from the particles, weighted by the particle weights:

$$E_{g\sim P(G|\tau)}[SS(g)] \approx \frac{\sum_i w_i SS(g^i)}{W}$$

$ESS(g,h_k,\tau_k)$ can be calculated for each submodel independently:

$$
\begin{aligned}
&E_{g,h_1,\ldots,h_N\sim P(G,H_1,\ldots,H_N|\tau)}[SS(g,h_k,\tau_k)] \\
&\qquad \approx \frac{1}{W}\sum_i w_i \int_{h_k} P(h_k|g^i,\tau_k)SS(g^i,h_k,\tau_k)dh_k \\
&\qquad = \frac{1}{W}\sum_i \frac{\prod_j P(\tau_j|g^i)}{P(\tau)} \int_{h_k} P(h_k|g^i,\tau_k)SS(g^i,h_k,\tau_k)dh_k \\
&\qquad \propto \frac{1}{W}\sum_i \prod_{j\neq k} P(\tau_j|g^i) \int_{h_k} P(h_k,\tau_k|g^i)SS(g^i,h_k,\tau_k)dh_k \\
&\qquad \propto \frac{1}{W}\sum_i w_i \int_{h_k} P(h_k,\tau_k|g^i)SS(g^i,h_k,\tau_k)dh_k
\end{aligned}
$$

```
**Function Wholemodel_Estep**
    **input**: current model $\theta^t$, evidence $\tau$
    **output**: Expected sufficient statistics $ESS$
    $ESS := \{ESS(g), ESS(s_1, g), \ldots, ESS(s_n, g)\}$
    Initialize $ESS$ as empty
    For each particle $g^i \in \{g^1, \ldots, g^M\}, g^i \sim P(G \mid \tau)$
        For each $S_j \in \{S_1, \ldots, S_N\}$
            $[P(\tau_j|g^i), ESS(s_j, g^i)] =$ **Submodel_Estep**$(g^i, \theta^t[S_j], \tau_j)$
        For each $S_j \in \{S_1, \ldots, S_N\}$
            $ESS(s_j, g) = ESS(s_j, g) + w_i \times ESS(s_j, g^i)$
        $ESS_{g^i} =$ **CountGSS**$(g^i)$
        $ESS(g) = ESS(g) + w_i \times ESS_{g^i}$
    Return $ESS$
```

Figure 4.7: Rao-Blackwellized particle filtering Estep for the whole model

where $w_i = \prod_{j \neq k} P(\tau_j|g^i)$.

$P(\tau_j|g^i)$ and $\int_{h_k} P(h_k, \tau_k|g^i) SS(g^i, h_k, \tau_k) dh_k$ can be calculated using the technique described in Chapter 3 for exact ESS calculation.

The full E-step algorithm is shown in Figure 4.7 (s_k represents all of the variables in submodel k). Function **Submodel_Estep** calculates the expected sufficient statistics and the likelihood for a subnet model. Since we sampled the full trajectory of G, we know exactly for each interval which conditional intensity matrix of the hidden variable H should be used, so a modified forward-backward exact inference algorithm for CTBNs can be used. Function **CountGSS** counts the empirical time and transition statistics from the sampled trajectory of G.

In EM, we use the ESS as if they were the true sufficient statistics to maximize the likelihood with respect to the parameters. For a "regular" CTBN variable X (such as our hidden variable G and H), the following equation performs the maximization.

$$Q_{X|u}(x, x') = \frac{\bar{M}[x, x'|u]}{\bar{T}[x|u]}$$

For our toggle variables, *e.g.* P_i, the maximization is

$$Q_{P_i|\mathbf{u}} = \frac{M_{P_i}}{T[U = \mathbf{u}]}$$

where M_{P_i} is the number of events for variable P_i and $Q_{P_i|\mathbf{u}}$ is the only parameter: the rate of switching.

We synchronize the particles at the end of each "window" and resample, as normal for a particle filter at those points.

4.3 Online Testing using Likelihood

Once the CTBN model has been fit to historic data, we detect attacks by computing the likelihood of a window of the data (see Section 5.1) under the model. If the likelihood falls below a threshold, we flag the window as anomalous. Otherwise, we mark it as normal.

In our experiments, we fix the window to be of a fixed time length, T_w. Therefore, if the window of interest starts at time T, we wish to calculate $p(\tau[T, T+T_w] \mid \tau[0, T])$ where $\tau[s, t]$ represents the joint trajectory of C_{inc}, C_{dec}, P_i, and P_o from s to t. In the factored model, this calculation involves integrating out the trajectories of U and H and can be done in an on-line fashion using the standard forward passing inference algorithm (Nodelman et al., 2002). In the connected model, we use a RBPF to estimate this probability.

Chapter 5

NIDS - Experiment Results

In this chapter, we present our experiment results of NIDS on real data using the approaches described in Chapter 4.

5.1 Datasets

We verify our approach on two publicly available real network traffic trace repositories: the MAWI working group backbone traffic (MAWI) and the LBNL/ICSI internal enterprise traffic (LBNL).

The MAWI backbone traffic is part of the WIDE project which has collected raw daily packet header traces since 2001. It records the network traffic through the inter-Pacific tunnel between Japan and the USA. The dataset uses `tcpdump` and IP anonymizing tools to record 15-minute traces every day, and consists mostly of traffic from or to Japanese universities. In our experiment, we use the traces from January 1st to 4th of 2008, with 36,592,148 connections over a total time of one hour.

The LBNL traces are recorded from a medium-sized site, with emphasis on characterizing internal enterprise traffic. Publicly released in an anonymized form, the LBNL data collects more than 100 hours network traces from thousands of internal hosts. From what is publicly released, we take one hour traces from January 7th, 2005 (the latest date available), with 3,665,018 total connections.

5.2 Worm Detection

We start with the problem of worm detection. For each of the datasets, we pick the ten most active hosts. To create a relatively abundant dataset based on the original packet traces, we constructed training-testing set pair for each IP address. We use the first half as a training set to learn the CTBN model. The other half we save for testing. Since the network data available are clean traffic with no known intrusions, we inject real attack traces into the testing data. In particular, we inject IP Scanner, W32.Mydoom, and Slammer. We then slide a fixed-time window over the testing traces, report a single log-likelihood value for each sliding window, and compare it with a predefined threshold. If it is below the threshold, we predict it as an abnormal time period. We define the ground truth for a window to be abnormal if any attack traffic exists in the interval, and normal otherwise. The window size we use is 50 seconds. We only consider windows that contain at least one network event.

When injecting the attack traffic, we randomly pick a starting point somewhere in the first half of the test trace and insert worm traffic for a duration equal to α times the length of the full testing trace. The shorter α is, the harder it is to detect the anomaly. We also scale back the rates of the worms. When running at full speed, a worm is easy to detect for any method. When it slows down (and thus blends into the background traffic better), it becomes more difficult to detect. We let β be the scaling rate (*e.g.* 0.1 indicates a worm running at one-tenth of its normal speed).

5.2.1 Results of the Factored Model

We compare our CTBN approach against the connection counting method, the nearest neighbor algorithm used in Lazarevic et al. (2003), and the adaptive naive Bayes approach of Agosta et al. (2007b).

The connection counting method is straightforward. We score a window by the number of initiated connections in the window. As most worms aggregate many connections in a short time, this method captures this particular anomaly well.

To make nearest neighbor competitive, we try to extract a reasonable set of features. We follow the feature selection of Lazarevic et al. (2003), who use a total of 23 features. Not all of their features are available in our data. Those available are shown in Figure 5.1. Notice that these features are associated with each connection record. To apply the nearest neighbor method to our window based testing framework, we first calculate the nearest distance of each connection inside the window to the training set (which is composed of normal traffic only), and assign the maximum among them as the score for the window, recall that high

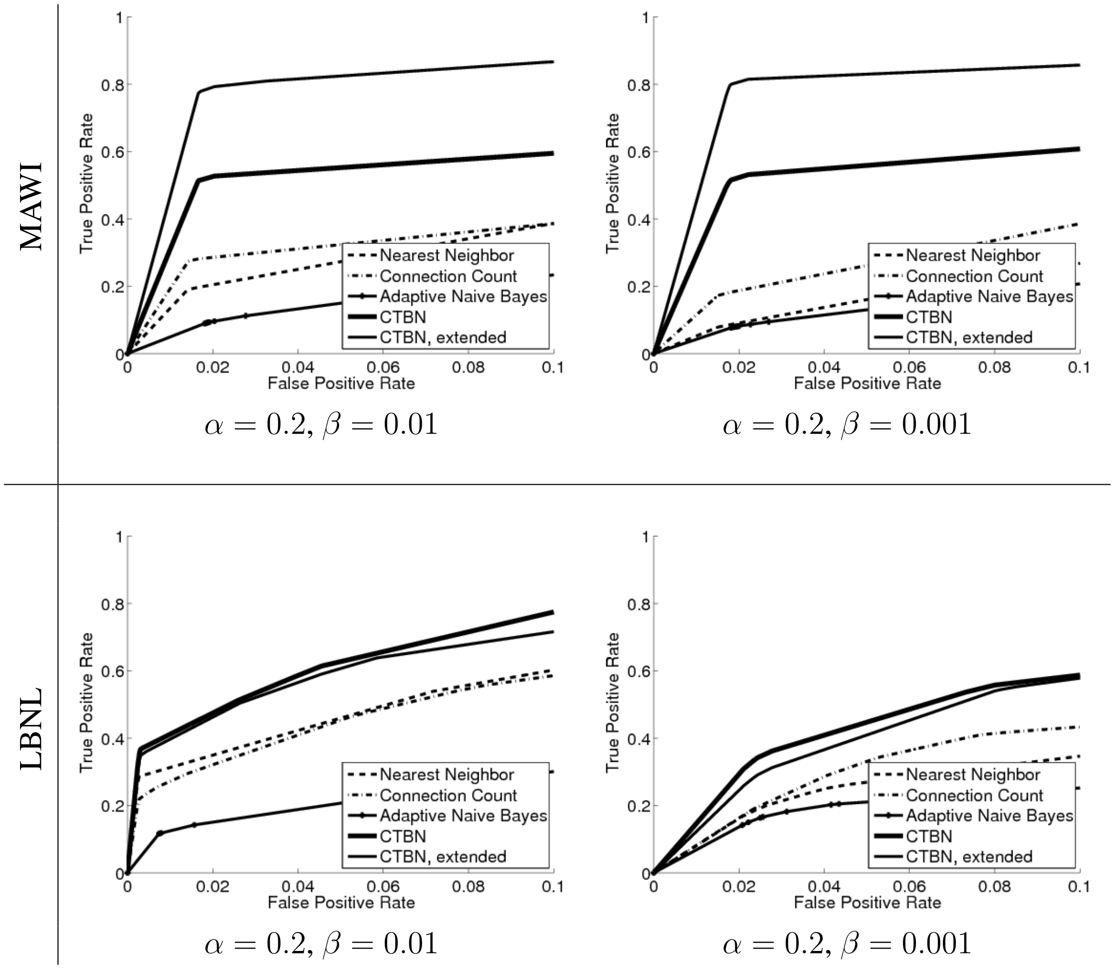
packets flowing from source to destination
packets flowing from destination to source
connections by the same source in the last 5 seconds
connections to the same destination in the last 5 seconds
different services from the same source in the last 5 seconds
different services to the same destination in the last 5 seconds
connections by the same source in the last 100 connections
connections to the same destination in the last 100 connections
connections with the same port and source in the last 100 connections
connections with the same port and destination in the last 100 connections

Figure 5.1: Features for nearest neighbor approach

scores indicate abnormality.

Finally, we employ the adaptive naive Bayes approach of Agosta et al. (2007b), which showed promising results on similar problems. We also follow the feature selection they presented, although not all of them are available in our dataset. To train the Naive Bayes network parameters, we use five available features: the number of new connections in the previous three windows and the entropy of the number of distinct destination IPs and ports in the current window. All the features are discretized into six evenly spaced bins. These features are not exactly the same as those from the nearest neighbor. Each method was tuned by the authors to work as well as possible, so we try to follow each of their methodologies as best as possible to give a fair comparison; this includes the selection of features.

Figure 5.2 compares the ROC curve for our method to those of the other methods for the IP scanning attack. The "CTBN" model is the one described in Section 6.2.2, where the hidden variable H and the mode variable U are two distinct variables in the network, while the "CTBN, extended" model is the one described in Section 4.1.3, where we allow



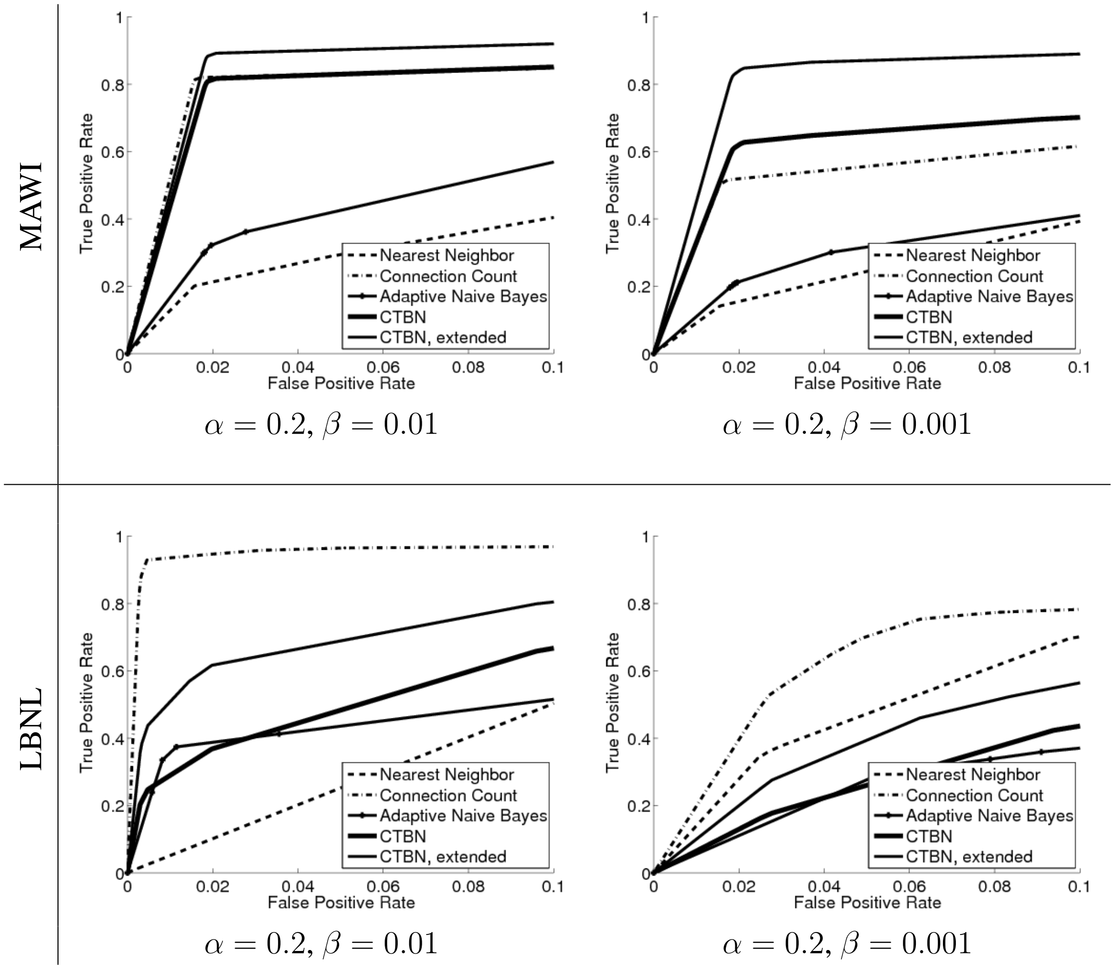
$\alpha = 0.2, \beta = 0.01$ $\alpha = 0.2, \beta = 0.001$

$\alpha = 0.2, \beta = 0.01$ $\alpha = 0.2, \beta = 0.001$

Figure 5.2: ROC curves of testing results on IP scanning attack. Top: MAWI. Bottom: LBNL.

simultaneous transitions for H and U. The curves show the overall performance on the 10 hosts we chose for each dataset. α represents the fraction of time during which the attack is present and β represents the speed of the attack. The curves demonstrate that as the attack becomes more subtle (β is smaller), our method performs relatively better compared with other methods. Figures 5.3 and 5.4 show the same curves but for the Mydoom and Slammer attacks.

Each point on the curve corresponds to a different threshold of the algorithm. Because



$\alpha = 0.2, \beta = 0.01$

$\alpha = 0.2, \beta = 0.001$

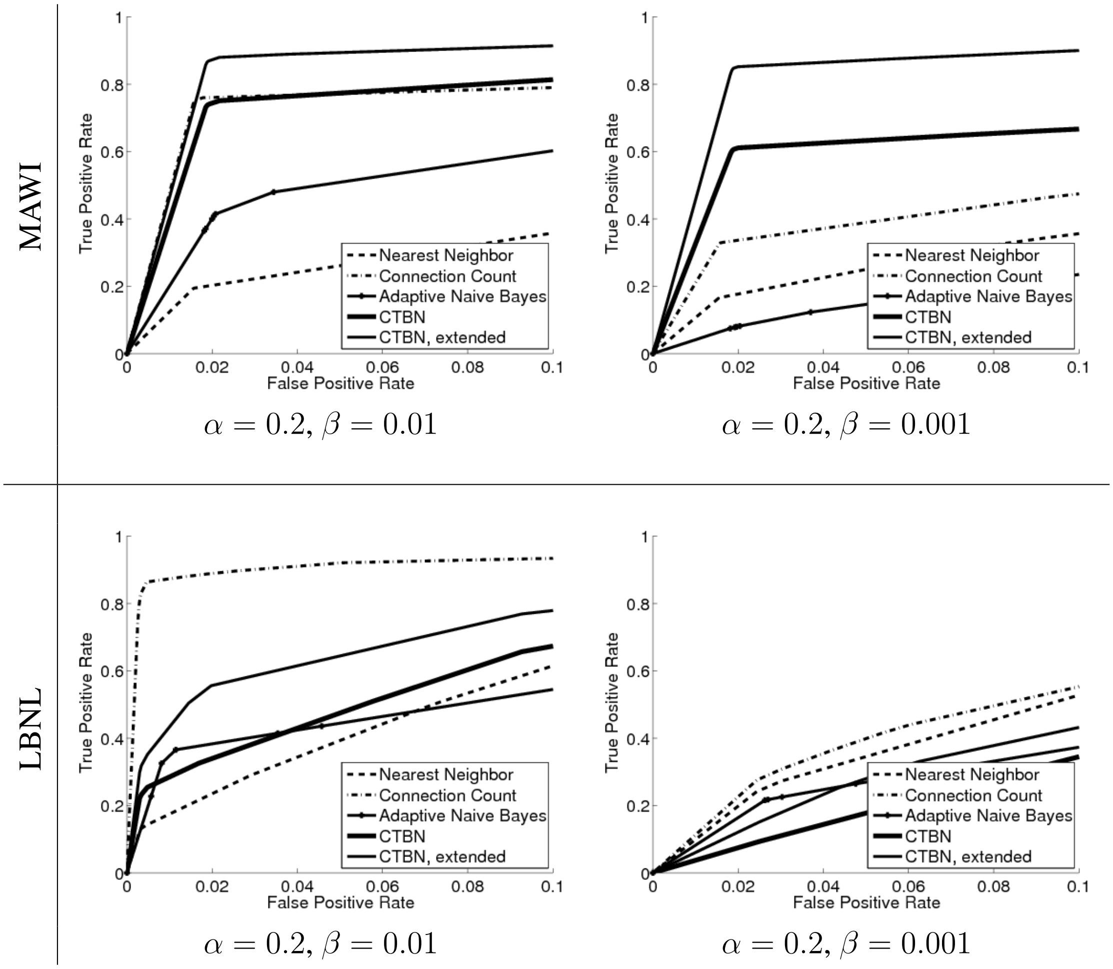
$\alpha = 0.2, \beta = 0.01$

$\alpha = 0.2, \beta = 0.001$

Figure 5.3: ROC curves of testing results on Mydoom attack. Top: MAWI. Bottom: LBNL.

attacks are relatively rare, compared to normal traffic, we are most interested in the region of the ROC curve with small false positive rates.

We note that our method out performs the other algorithms consistently for the MAWI dataset. For the LBNL dataset, with the Mydoom and Slammer attacks, some of the other methods have better performance, in particular a simple connection counting method performs the best. We are uncertain of the exact reason, but suspect it may be due to differences in the traffic type (in particular, the LBNL data comes from enterprise traffic). The addition of



$\alpha = 0.2, \beta = 0.01$

$\alpha = 0.2, \beta = 0.001$

$\alpha = 0.2, \beta = 0.01$

$\alpha = 0.2, \beta = 0.001$

Figure 5.4: ROC curves of testing results on Slammer attack. Top: MAWI. Bottom: LBNL.

a hidden variable to our model allows us to model non-exponential durations. However, the complexity of such a phase-type distribution depends on the number of states in the hidden variable. If there are not enough states to model the true duration distribution, the algorithm may end up with an exponential model (at least for some events). Exponential models do not disfavor (in terms of likelihood) many quick transitions, compared to heavier tailed distributions. This might lead to worse performance in exactly the same situations where a connection-count method would work well.

5.2.2 Results of the Connected Model

We notice that in the experimental results for the factored model, adaptive Naive Bayes algorithm is outperformed by the other methods in most cases. So we drop this method. We compare our method employing RBPF with our previous factored CTBN model, connection counting, nearest neighbor, Parzen-window detector (Yeung and Chow, 2002), and one-class SVM with a spectrum string kernel (Leslie et al., 2002).

We use the same method for connection counting and nearest neighbor as described in 5.2.1. For the Parzen window approach, we apply the same feature set as for nearest neighbor method, and assign the minimum density among all the connections inside a window to be the score of that window.

Besides the above feature-based algorithms, we would also like to see how sequence-based approaches compare against our methods. They are widely used in network anomaly detection. Like our approach, they treat the traffic traces as stream data so that sequential contexts can be explored. One-class SVM with spectrum string kernel was chosen for comparison. We implemented a spectrum kernel in the LIBSVM library. We give the network activities (such as a connection starting or ending, or a packet emmision or receipt) inside each port-level submodel a distinct symbol. The sequence of these symbols are fed to the algorithm as inputs. A decision surface is trained from normal training traffic. In testing, for each sliding window, the distance from this window string to the decision hyperplane is reported as the window score. We also tried experiments using the edit distance kernel, but their results are dominated by the spectrum kernel, so we do not report them here.

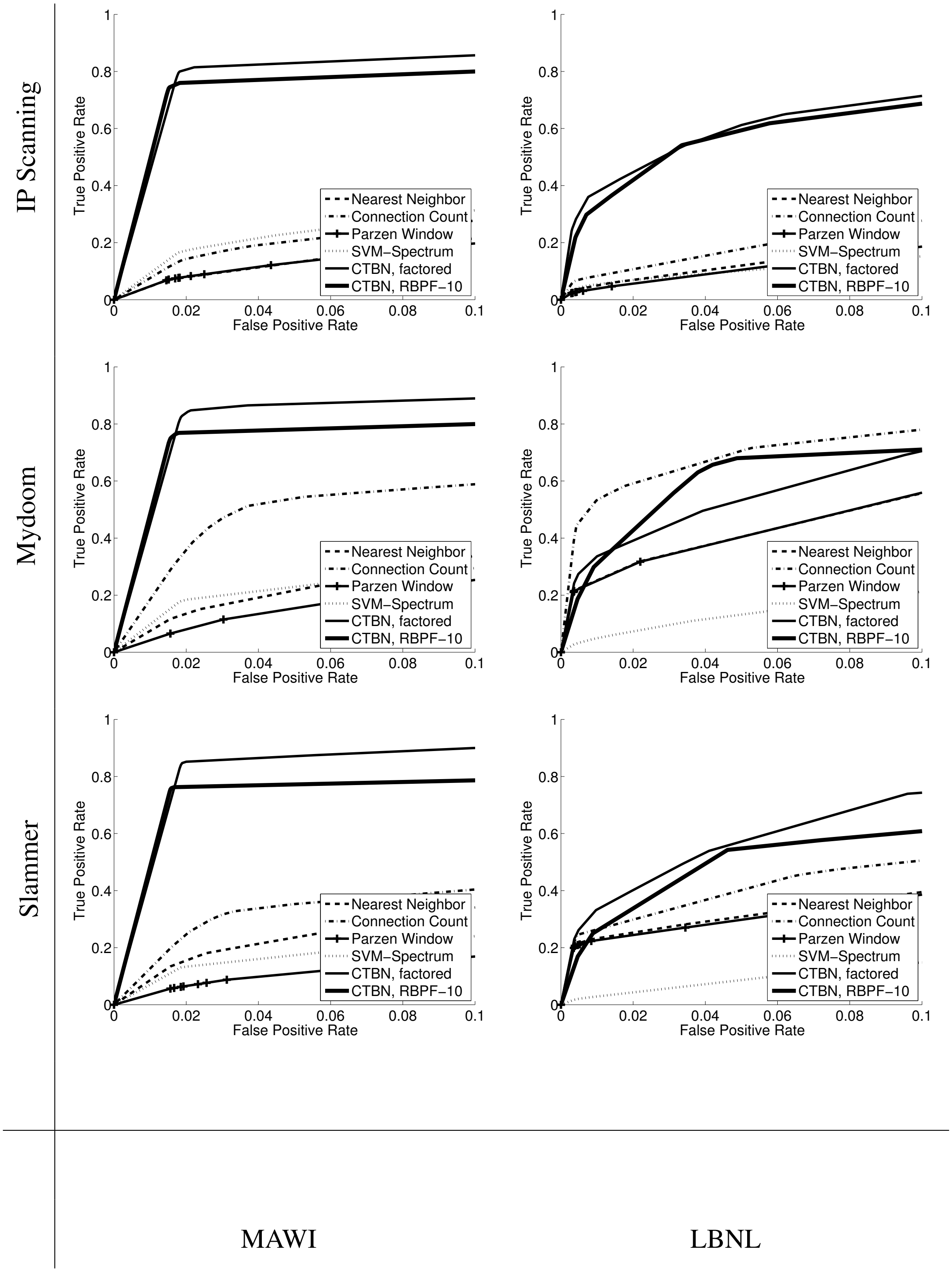


Figure 5.5: ROC curves of testing results on IP scanning attack, Mydoom attack and Slammer attack. $\beta = 0.001$. Top: MAWI. Bottom: LBNL.

For our method, we set the state space of variable G to be 4 and variable H to be 8. We use 10 samples for particle filtering, and resample the particles after every 50 seconds. For the SVM spectrum kernel method, we choose the sub-sequence length to be 5 and the parameter ν to be 0.8. We set α to be 0.02 for all the experiments here to challenge the detection tasks.

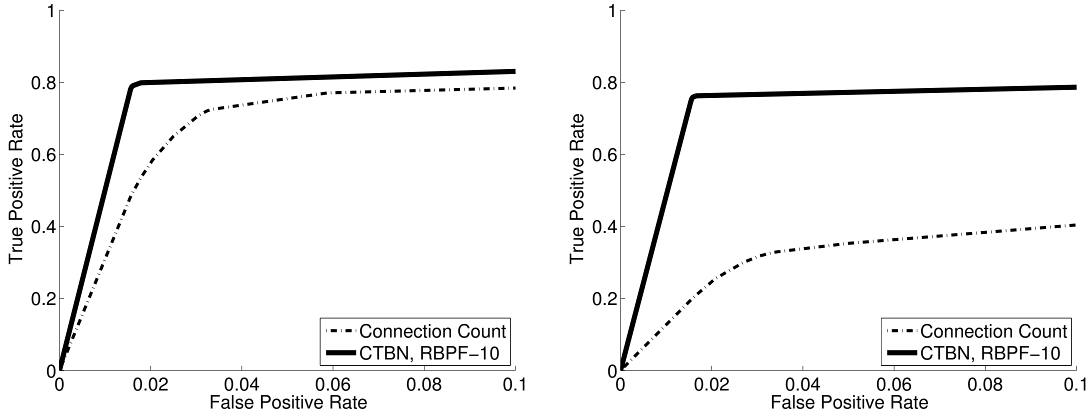


Figure 5.6: ROC curves of testing results for Slammer attack on MAWI dataset demonstrating the effect of slowing the attack rate. Left: $\beta = 0.01$. Right: $\beta = 0.001$

We show the ROC curves of all the methods in Figure 5.5. The curves show the overall performance on the 10 most active hosts for each dataset. Each point on the curves corresponds to a different threshold of the algorithm. Our CTBN method, both the fully connected model and the factored model, out-performs the other algorithms except in the single case of the Mydoom attack against a background of the LBNL traffic. In many cases, the advantages of the CTBN approach are pronounced. The connected model is outperformed by the factored model when the false positive rate goes higher. But if we restrict the false positive rate to be lower. i.e. less than 0.01, the connected model is more accurate. In real-world detection, a small false positive rate is usually preferred.

We also show how the ROC curves shift if we scale back the worm running speed β in Figure 5.6. As firewalls are built to be more sensitive to block malicious traffic, worms have to act more stealthy to sneak through. We demonstrate the robustness of our method compared to the best competitor (connection counts) to the speed of the worm's attack.

5.3 Host Identification

Identifying individual hosts based on their network traffic patterns is another useful application of our model. For instance, a household usually installs a network router. Each family member's computer is connected to this router. To the outside Internet, the network traffic going out of the router behaves as if it is coming from one peer, but it is actually coming from different people. Dad will possibly read sports news while kids surf on social networks. It is interesting as well as useful to tell which family member is contributing the current network traffic. Host identification can also be used to combat identity theft. When a network identity is abused by the attacker, host identification techniques can help the network administrator tell whether the current network traffic of this host is consistent with its usual pattern or not.

Host	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.09	0.41	**0.47**	0	0	0	0	0.03	0	0
2	0.06	**0.50**	0.31	0	0	0.13	0	0	0	0
3	0	0	**1**	0	0	0	0	0	0	0
4	0	0	0	**1**	0	0	0	0	0	0
5	0	0.08	0.13	0	**0.68**	0.06	0.03	0.01	0	0.01
6	0	0.20	0.03	0	0.01	**0.74**	0	0.02	0	0
7	0.25	0	0.06	0.02	0	0	**0.66**	0	0	0.01
8	0	0.08	0.28	0	0	0.05	0	**0.59**	0	0
9	0.04	0.05	0.13	0.01	0.01	0.01	0.02	0	**0.73**	0
10	0	0.03	0.18	0.01	0	0.15	0.03	0.03	0	**0.57**

Figure 5.7: Confusion matrix for LBNL for host identification using CTBN

The first set of experiments we construct is a host model fitting competition. The same 10 hosts picked for the worm detection tasks from LBNL dataset compose our testing pool. We learn the coupled CTBN model for each host. We split the test traces (clean) of a particular host into segments with lengths of 15 seconds. For each of the segments, we compute the log-likelihood of the segment under the learned model from all the hosts (including its own), and label the segment with the host that achieves the highest value. We compute a confusion matrix C whose element C_{ij} equals the fraction of test traces of host i for which model j has highest log-likelihood. We expect to see the highest hit rates fall on the diagonals because ideally a host should be best described by its own model. Table 5.7 shows our results on the dataset of LBNL. The vast majority of traffic windows are assigned to the correct host. With the exception of host 1, the diagonals are distinctly higher than other elements in the same row. For comparison, we performed the same experiment using SVM spectrum kernel method. Again, we selected the sub-sequence length to be 5 and the parameter ν to be 0.8. We tried multiple methods for normalization (of the distance to the hyperplane) and variations of parameters. All produced very poor results with almost all of the windows assigned to a single host. The results are shown in Table 5.8.

Host	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**0.93**	0	0	0	0	0	0	0	0	0.07
2	**0.57**	0	0	0	0	0	0	0	0	0.43
3	**0.5**	0	0	0	0	0	0	0	0	**0.5**
4	**0.5**	0	0	0	0	0	0	0	0	**0.5**
5	**0.52**	0	0	0	0	0	0	0	0	0.48
6	0.36	0	0	0	0	0	0	0	0	**0.64**
7	**0.59**	0	0.08	0	0	0	0	0	0	0.33
8	0.44	0	0	0	0	0	0	0	0	**0.56**
9	**0.59**	0	0	0	0	0	0	0	0	0.41
10	**0.51**	0	0	0	0	0	0	0	0	0.49

Figure 5.8: Confusion matrix for LBNL for host identification using SVM Spectrum kernel

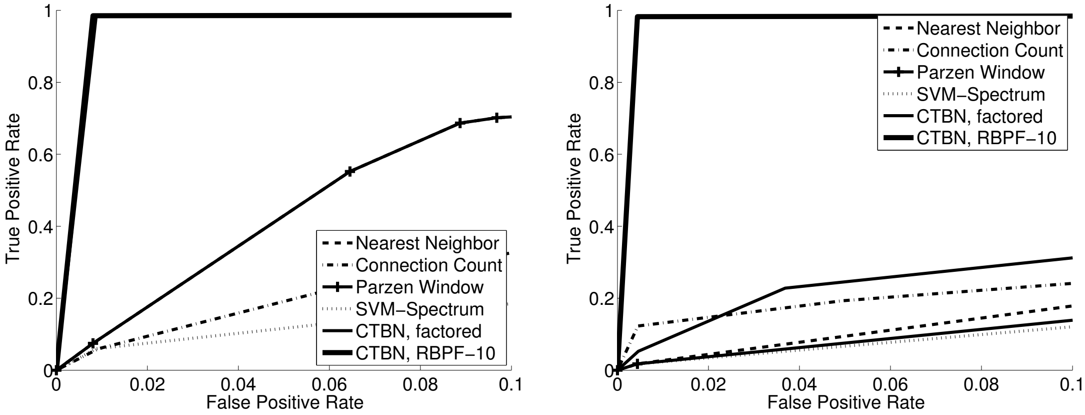


Figure 5.9: ROC curves of testing results on host identification on the LBNL data. Left: host 1, Nearest neighbor curve and Parzen window curve overlap, both CTBN curves overlap. Right: host 2.

Our second experiment is a host traffic differentiation task. We mingle the network traffic from another host with the analyzed host. We expect the detection method to successfully tell apart the two. To verify this idea, we pick one host among the 10 we choose above from LBNL dataset and split its traffic evenly into training and testing. We again learn the model from training data. For testing data, we randomly choose a period and inject another host's traffic as if it were a worm. Our goal is to identify the period as abnormal since the host's traffic is no longer its own behavior. Figure 5.9 displays the results from two such combination tests. The parameters for injecting the traffic as a worm are $\alpha = 0.02$, $\beta = 0.001$. In the left graph, the nearest neighbor and Parzen window curve overlap, and both CTBN curves overlap. In the right graph, the coupled CTBN curve substantially outperforms all the other curves.

Chapter 6

Extensions

In addition to the work describe in Chapter 4, we also extend our NIDS in several aspects. In this chapter, we present an approximation method to speed up the detection in Section 6.1. We also show another NIDS detector using CTBNs in Section 6.2.

6.1 Speed Up

In Section 4.1, we present a factored CTBN model for network traffic. We achieve good performance in intrusion detection on real datasets using this model. However, it still faces a problem: the computational efficiency. Exact inference is used while learning the model in the training phase and calculating the likelihood in the testing phase. Although it could take a while to estimate the model parameters in the training phase, the models are learned once offline and stored for further usage. So time is not a critical issue. But in the testing phase, we expect to receive prompt output from the algorithm so that the detection can be made in time. The longer the prediction takes, the more destruction it could make to the computer system. The current exact inference method used for calculating the likelihood requires a significant number of computations of matrix operations. They usually take a great amount of time. Therefore, to make our detector efficient and realistic, we develop an approximate likelihood calculation algorithm to speed up the detection.

6.1.1 Approximate Likelihood Calculation

In the testing phase, the likelihood of a window of the data under the model is calculated. Assume we have a trajectory τ beginning at time 0 and ending at time T. Events happen at time stamps $0, ..., t_i, t_{i+1}, ..., T$. We define the vector $\alpha_t = p(X_t, \tau_{0:t^+})$ to be the distribution over the state (of the hidden variables) given the trajectory until time t. The likelihood is computed as the summation over all the enties in the vector α_T.

To calculate α_T, we use a forward pass filtering algorithm to propagate αs over the entire trajectory at every time stamp t_i. The propagation rule over the interval $[t_i, t_{i+1}]$ is

$$\alpha_{t_{i+1}} = \alpha_{t_i} \exp^{Q_{S_i}(t_{i+1}-t_i)} Q_{S_i S_{i+1}},$$

where Q_{S_i} and $Q_{S_i S_{i+1}}$ are defined from the parameters of the model and the observed data (see Section 3.2).

The computation of $\alpha_{t_i} \exp^{Q_{S_i}(t_{i+1}-t_i)}$ involves the matrix exponential. It is currently performed by the Runge-Kutta ODE method of fifth order (Asmussen et al., 1996). This method integrates over small steps whose sizes are adaptive to the interval $[t_i, t_{i+1}]$. The time complexity of the method highly depends on the number of steps, which is a function of the step size. The larger the intensities of the transition matrix Q_{S_i} is, the smaller the step size is needed to catch up with the fast changing rate. Inside each step, a constant number of matrix operations are computed. In our case, network activities take place frequently. The intensities of the transition matrix are relatively high. This indicates a smaller step size.

Instead of using the Runge-Kutta method, we present a different way to calculate the above function $\alpha_{t_i} \exp^{Q_{S_i}(t_{i+1}-t_i)}$.

Say $t = t_{i+1} - t_i$. We first pick up a small Δt.

When $t > \Delta t$, let $m = \lfloor \frac{t}{\Delta t} \rfloor$ be the number of Δts within the interval. We have

$$\exp^{Q_{S_i} t} \approx \exp^{Q_{S_i} m \Delta t} = \exp^{Q_{S_i} b_0 2^0 \Delta t} \times \exp^{Q_{S_i} b_1 2^1 \Delta t} \times \exp^{Q_{S_i} b_2 2^2 \Delta t} \ldots,$$

where b_i is the ith binary digit in the representation of m. We pre-compute $\exp^{Q_{S_i} \Delta t}$, $\exp^{Q_{S_i} 2 \Delta t}$, $\exp^{Q_{S_i} 4 \Delta t}$, ... Then $\exp^{Q_{S_i} m \Delta t}$ can be simply calculated by multiplications of these pre-computed matrix exponentials.

When $t < \Delta t$, $\exp^{Q_{S_i} t}$ can be approximated by its first order Taylor expansion:

$$\exp^{Q_{S_i} t} \approx I + Q_{S_i} t,$$

when Δt is small.

6.1.2 Experiment Results

We use the same datasets and experiment set up as described in Section 5.2.1. We verify our approximation likelihood calculation method on the task of worm detection. For the MAWI dataset and the LBNL dataset, ten most active hosts are picked. For each of them, the trace (clean traffic only) is split into training-testing pairs. Worms are injected into the testing data. A model is learned from the training data. In testing, the likelihood of a window of the data is assigned as the score of the window. It is compared to a predefined threshold to trigger the alarm.

Figure 4.5 shows the port-level submodels we used in this experiment. The worm we used is IP Scanning worm. We test our algorithm on both datasets. We recalculate the likelihood of the testing data using the above approximation method. In specific, we choose Δt to be $0.1 \times \frac{1}{Q_{diagmax}}$, where $Q_{diagmax}$ is the maximum absolute value among the diagonal intensities of the transition matrix Q. $\frac{1}{Q_{diagmax}}$ shows the finest time granularity of the system transitions. We therefore need Δt to be smaller than this.

For comparison, we plot the ROC curves to show the detection accuracy as we did before. The right graphs of Figure 6.1 and Figure 6.3 show the resulting plots on the MAWI dataset and LBNL dataset respectively. The plots show that the approximation methods achieves similar performance to the exact method on both datasets.

To make a more straightforward comparison of the two methods, we also generate the cross plots for the outputs (the likelihoods of the windows) from the two algorithms. They are shown in the right graphs of Figure 6.1 and Figure 6.3. For any window, we expect to see

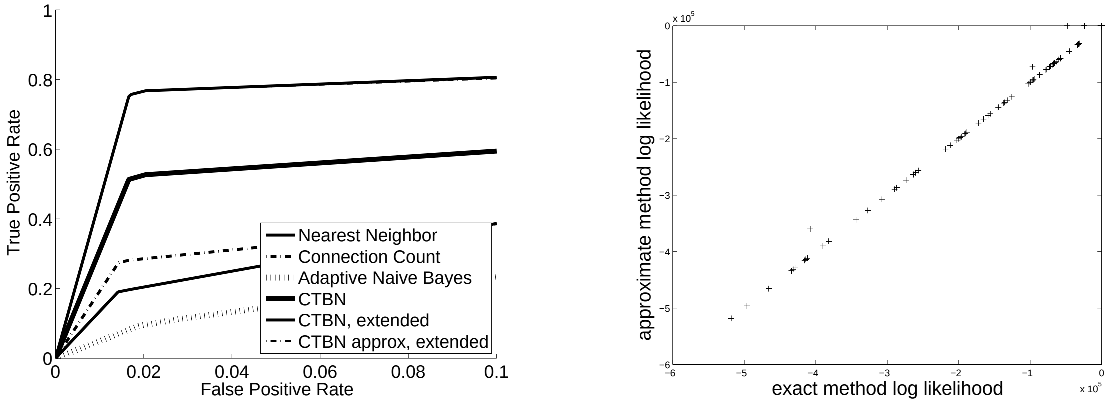


Figure 6.1: Left: ROC curves of testing results for IP Scanning attack on MAWI dataset demonstrating the effect of speed up. The "CTBN, extended" curve employs the factored model containing a single hidden variable H, and the "CTBN approx, extended" curve uses approximate likelihood calculation on the "CTBN, extended" model. Left: The cross plot of log likelihood calculated via the exact method and the approximate method

the likelihood calculated by the approximation method is close to the true value computed by the exact method. The two cross plots show that the approximation method has met this expectation.

Since our motivation is to speed up the detection, we are highly interested in how much the approximation method accelerates the detection task. Therefore, we compare the time efficiency in addition to the accuracy. We record the running time of both methods. Table 6.2 and Table 6.4 list the detection times of both methods. The test data on the MAWI dataset

Host	Exact method time (seconds)	Approximate method time (seconds)
1	212.024	22.784
2	297.808	28.376
3	1183.560	57.540
4	95.228	14.520
5	335.652	25.640
6	258.136	25.448
7	237.968	20.200
8	163.612	40.576
9	513.682	61.964
10	182.896	32.184

Figure 6.2: MAWI dataset, IP Scanning worm, Running time of exact inference method and approximation method

lasts about 480 seconds. The exact method takes a running time less than that in most cases, which is good. But the approximation is about 10 times faster. For the LBNL dataset, the test data lasts about 2400 seconds. The exact method runs at various speeds on different hosts, but can take as long as 78 hours for one host. This is not acceptable. The approximation method in this case accelerates the computation to a few seconds. Even for the toughest host, it finishes in less than 2 minutes.

6.2 Another NIDS using CTBNs

In Chapter 4, we describe our work of NIDS, where the network packet header traces are modeled using factored or connected CTBN model, and future traffic within a time window are flagged according to its likelihood under the model. In this section, we present another NIDS developed. This NIDS is different from the previous ones in that the port-level submodel contains a counting variable. Also, in the testing phase, a Kullback-Leibler

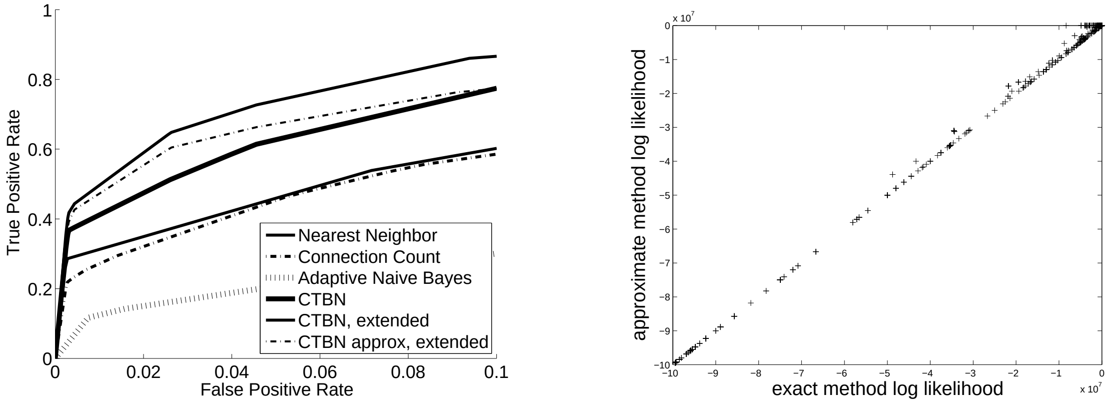


Figure 6.3: Left: ROC curves of testing results for IP Scanning attack on LBNL dataset demonstrating the effect of speed up. Left: The cross plot of log likelihood calculated via the exact method and the approximate method

(KL-)divergence is used to calculate the deviation from the norm.

6.2.1 CTBN Model for Network Traffic

Figure 6.5 shows the most popular ports ranked by their frequencies in the network traffic on our dataset (see Section 6.3).

We make the same assumption as in Chapter 4.1 that network traffic are independent across the destination ports. Besides, we observe that packet activities in network traffic are highly dependent on the host's connection status. The more concurrent connections there are,

Host	Exact method time (seconds)	Approximate method time (seconds)
1	281789.322	115.004
2	127011.111	1.768
3	0.931	0.476
4	383.892	1.728
5	139555.273	45.260
6	762.138	0.968
7	86.112	0.188
8	45784.594	1.868
9	110237.578	3.076
10	0.321	0.144

Figure 6.4: LBNL dataset, IP Scanning worm, Running time of exact inference method and approximation method

PORT	DESCRIPTION
53	Domain Name Server
88	Kerberos
80	World Wide Web HTTP
389	Lightweight Directory Access Protocol
445	Microsoft-DS
911	xact-backup
135	DCE endpoint resolution
139	NETBIOS Session Service

Figure 6.5: Ranking of the most frequent ports on our Intel dataset.

the more frequently packets are sent or received. While the arrival times for events across an entire CTBN is distributed exponentially (as the entire system is Markovian), once certain variables are unobserved, the marginal distribution over the remaining variables is no longer Markovian.

Our port-level submodel is composed of a concurrent connection count variable (C), a packet-in (P_i) variable and a packet-out (P_o) variable as C's children, and a latent variable (H) as a parent of the other three. This structure is shown in Figure 6.2.1. The experimental results of this NIDS are for binary hidden variables.

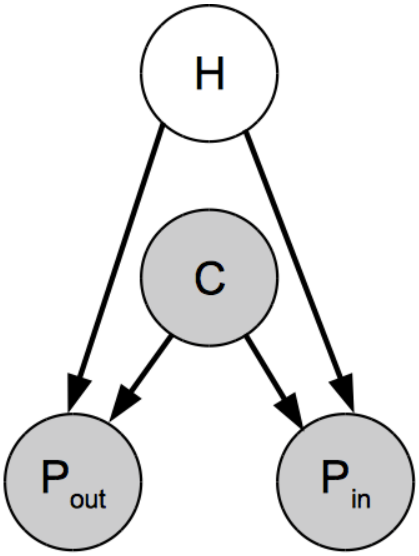


Figure 6.6: Individual port-level submodel

6.2.2 Adding Counting Variables to CTBNs

The concurrent connection count variable, C, poses a slight modeling problem for CTBNs. As originally presented, CTBNs can only deal with finite-domain variables. The number of concurrent connections can potentially grow without bound. Our traffic examples do exhibit a wide range of concurrent connection counts. We tried quantizing C into fixed bins, but the results were fairly poor. We instead note that C can only increase or decrease by one at any given event (the beginning or ending time of a connection). Furthermore, we make the assumption that the arrival of a connection is independent of the number of other connections, but that the termination of a connection is dependent on the current number of connections. In particular, we assume each current connection has the same intensity of stopping. This implies that the intensity with which *some* connection stops is proportional to the number of concurrent connections. C is thus a random (biased) walk constrained to the non-negative integers.

Let Q_{inc} be the intensity for the arrival of a new connection. Given the current concurrent connection count c, the intensity for decreasing $Q_{dec|c}$ is simply $c\tilde{Q}_{dec}$, where $\tilde{Q}_{dec}$ is the rate of the termination of a connection. Let $Q_{change|c} = Q_{dec|c} + Q_{inc}$. The resulting intensity matrix has the form

$$\mathbf{Q_{C|h}} = \begin{pmatrix} & \ddots & & & & & \\ 0 & Q_{dec|i-1} & -Q_{change|i-1} & Q_{inc} & 0 & \cdots & \\ \cdots & 0 & Q_{dec|i} & -Q_{change|i} & Q_{inc} & 0 & \cdots \\ & \cdots & 0 & Q_{dec|i+1} & -Q_{change|i+1} & Q_{inc} & 0 \\ & & & & & \ddots & \end{pmatrix}.$$

Note that there is one of these matrices (with its own $\tilde{Q}_{dec}$ and Q_{inc}) for each value of the hidden variable.

The variables P_i and P_o are children of C and therefore also require special consideration. Their parents have an infinite number of states, and thus we cannot store one intensity matrix for each parent value. However, the packet rate in network traffic is usually proportional to the concurrent connection count. We therefore also fix the structure of the CIM for P_i and P_o:

$$Q_{P_i|c,h} = c\tilde{Q}_{P_i|h}$$

$$Q_{P_o|c,h} = c\tilde{Q}_{P_o|h},$$

where $\tilde{Q}$ is a fixed rate matrix and c is the current value of the counting variable.

6.2.3 Parameter Estimation

We use the expectation maximization (EM) algorithm to estimate the parameters (Nodelman et al., 2005b).

For our new types of variables (counting variables and their children), we need to derive different sufficient statistics. Let C be a counting variable in a CTBN, and $\mathbf{U}$ be the parents of C. For C, the ESS we need are $\bar{M}_C[inc|\mathbf{u}]$ and $\bar{M}_C[dec|\mathbf{u}]$: the expected number of times C increases (or decreases) by 1 conditioned on its parent instantiation $\mathbf{u}$. We also need the total sum of all connection durations. Broken up by the value of its parent ($\mathbf{u}$), this becomes $\bar{R}_{dec|\mathbf{u}} = \sum_c c\bar{T}_{c|\mathbf{u}}$ where c is the number of concurrent connections and $\bar{T}_{c|\mathbf{u}}$ is the total expected amount of time this number of connections occurred while $\mathbf{U} = \mathbf{u}$. In our network, $\mathbf{U} = H$.

For any variable Z that has C as its parent, let $\mathbf{V}$ be Z's parents other than C. The "time" ESS for Z is now $\bar{R}_{Z|\mathbf{V}}[\mathbf{v}] = \sum_c c\bar{T}_{Z|\mathbf{V}}[Z|c, \mathbf{v}]$, where c is the instantiation of the counting variable C and $\mathbf{v}$ is the instantiation of $\mathbf{V}$. Z's expected number of transitions, $\bar{M}_{Z|\mathbf{V}}[z, z'|\mathbf{v}]$, can be calculated as for a regular CTBN variable.

In EM, we use the ESS as if they were the true sufficient statistics to maximize the likelihood with respect to the parameters. For a "regular" CTBN variable X (such as our hidden variable H), the following equation performs the maximization.

$$Q_X(x, x') = \frac{\bar{M}_X[x, x']}{\bar{T}_X[x]}$$

For our new counting variable, C, the maximizations are

$$Q_{inc|\mathbf{u}} = \frac{\bar{M}_C[inc|\mathbf{u}]}{T} \qquad\qquad \tilde{Q}_{dec|\mathbf{u}} = \frac{\bar{M}_C[dec|\mathbf{u}]}{\bar{R}_{dec|\mathbf{u}}} \ .$$

Here T is the total time of the trajectory. And for Z, a child of C with other parents $\mathbf{V}$, the maximization is

$$Q_{Z|\mathbf{V}}(z, z') = \frac{M_{Z|\mathbf{V}}[z, z'|\mathbf{v}]}{\bar{R}_{Z|\mathbf{V}}[\mathbf{v}]} \ .$$

Because only H is hidden in our model, the above sufficient statistics can be calculated using the exact inference algorithm of Nodelman et al. (2002).

6.2.4 KL-divergence Approximation

We fit the CTBN model to historic data. To detect the attacks, we must define a measure of the traffic's deviation from the CTBN model. When faced with a new stream of data, we do not query the model separately for each event to find out the new datum's likelihood under the model. CTBNs define distributions over trajectories (sequences of events) and not individual connections or packets.

We calculate the KL-divergence between the model and the testing distribution. If we let

P be the learned CTBN distribution for normal traffic, and Q be the distribution of the testing traffic, then the KL-divergence can be written as $D(Q\|P) = -H(Q) - E_Q \log P(D)$, where H is entropy. If we take Q to be the empirical distribution from a single trajectory, then the second term is merely the log likelihood of the trajectory under the CTBN model. This can be calculated using the standard inference algorithm.

However, it is relatively hard to compute the first term directly since a testing trajectory is just a single sample from Q. Thus we instead approximate the entropy by making a few assumptions. Note that these assumptions differ from those made to estimate the second term and therefore the resulting divergence estimate may be negative.

First, we assume that the model factors according to the destination ports (just as our CTBN model does). Therefore, the total entropy is the sum of the entropy of the traffic on each port. Second, we assume the sample was drawn from a semi-Markov process. That is, the time between transitions is non-exponential, but the embedded transition chain is Markovian. We flatten the visible variables (C, P_i, and P_o) in one port submodel into a single large variable. We estimate the entropy of the embedded Markov chain by using the empirical distribution of transitions.

For the distribution of time between transitions, we use a non-parametric estimator based on m-spacings (and, in particular, choose $m = \sqrt{n}$, where n is the number of transitions) to approximate the entropy. See Beirlant et al. (1997) for more details. The resulting estimator is

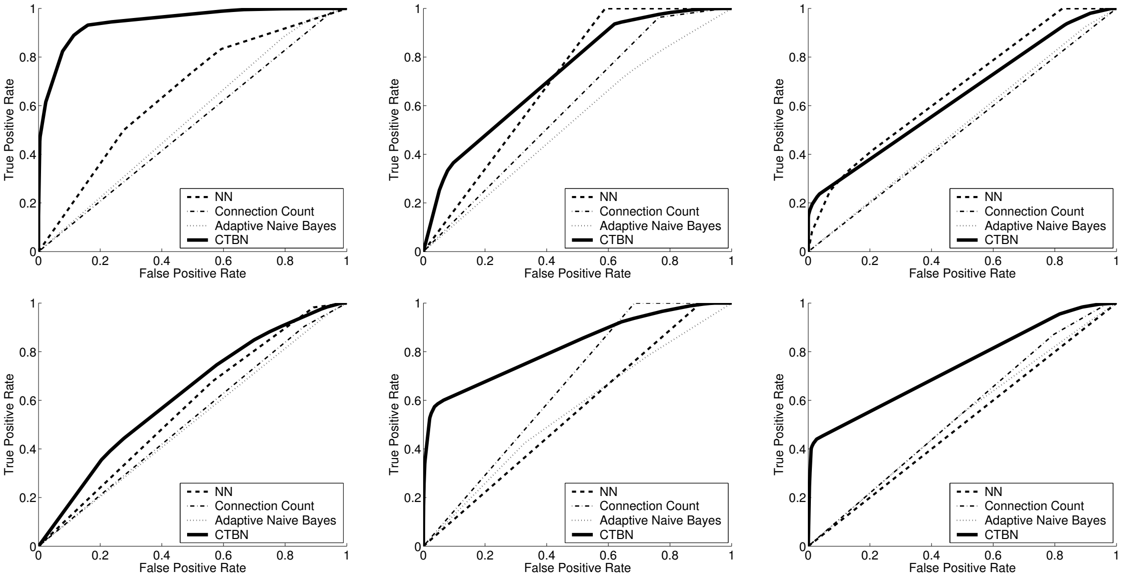


Figure 6.7: ROC curves for 6 hosts

$$\hat{H}_{\text{m-spacing}}(T^1, .., T^N) = \frac{m}{N-1} \sum_{i=0}^{\frac{N-1}{m}-1} \log\left(\frac{N+1}{m}(T^{(m(i+1)+1)} - T^{(mi+1)})\right)$$

where $T^{(i)}$ is the ith shortest transition duration. That is, we take the transition durations, sort them, and then re-index them according to their sorted order. This is an unbiased estimate of the entropy of the empirical distribution that makes no parametric assumptions on the form of the data.

6.3 Experiment Results

We verify our approach on “The Forbidden City” dataset provided by Intel Labs. The dataset contains 37 individual real machine traces. All the traces are normal. We randomly picked 6 hosts from the entire network. To create a relatively abundant dataset based on the original packet traces, we constructed a training-testing set pair for each IP address. We use half of the traces as a training set to learn the CTBN model for a given IP (host), and use a worm simulator (NLANR) to inject IP-scan worm activity into the remaining half to make our testing set.

For each algorithm (our CTBN algorithm and those below), we computed its score on consecutive non-overlapping windows of 50 transitions in the testing set. If the score exceeds a threshold, we declare the window a positive example of a threat. We evaluate the algorithm by comparing to the true answer: a window is a positive example if at least one worm connection exists in the window.

We compare against the nearest neighbor algorithms used in Lazarevic et al. (2003). Not all of the features in Lazarevic et al. (2003) are available. The features available in our dataset are

(1) the number of connections made by the same source as the current connection in the last 5 seconds, (2) the number of different services from the same source as the current connection in the last 5 seconds, and (3) the number of connections with the same services made by the same source as the current connection in the last 100 connections.

We also compare with a connection counting method. As IP-scan worms aggregate many connections in a short time, this method captures this particular anomaly well. We score a window by the average rate of connections in the window. While a connection count might not detect other kinds of attacks, it is a good base-line for this particular attack. Finally, we implemented the adaptive naive Bayes approach of Agosta et al. (2007b) which was designed for this same traffic dataset.

In Figure 6.7 we plot ROC curves for the 6 hosts. In these plots, we scaled back the worm injection rate to 10% of its normal rate. When running at full speed, the worm is easy to detect for any method. When it slows down (and thus blends into the background traffic better), it becomes more difficult to detect. Each point on the curve corresponds to a different threshold of the algorithm. Because attacks are relatively rare, compared to normal traffic, we are most interested in the region of the ROC curve with small false positive rates. We note that while the curves are drastically different for each host (representing the differences in normal background traffic), our method consistently performs better than the other techniques.

Chapter 7

HIDS

We can adopt the previous method to other data easily. Now we turn to the problem of detecting anomalies using system call logs.

7.1 A CTBN Model for System Calls

System call logs monitor the kernel activities of machines. They record detailed information of the sequence of system calls to operating system. Many malicious attacks on the host can be revealed directly from the internal logs.

We analyze the audit log format of SUN's Solaris Basic Security Module (BSM) praudit audit logs. Each user-level and kernel event record has at least three tokens: header, subject, and return. An event begins with a "header" in the format of: header, record length in bytes, audit record version number, event description, event description modifier, time and date. The "subject" line consists of: subject, user audit ID, effective user ID, effective group ID,

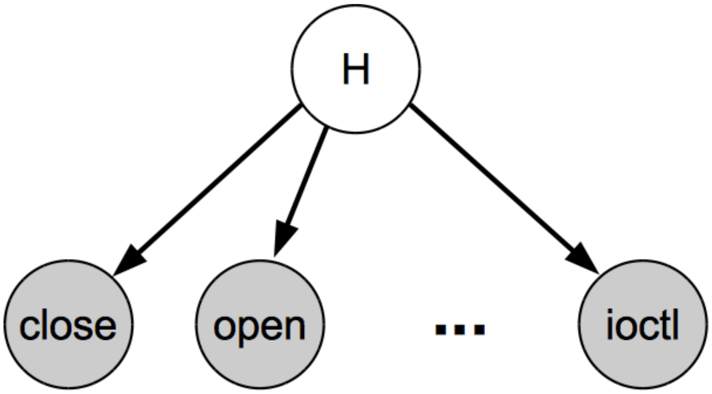


Figure 7.1: CTBN model for system call data

real user ID, real group ID, process ID, session ID, and terminal ID consisting of a device and machine name. A "return" with a return value indicating the success of the event closes the record.

We construct a CTBN model similar to our port-level network model. Individual system calls $S_1, ..., S_N$, which are the event description fields in the header token, are transiently observed: they happen instantaneously with no duration. We treat them as toggle variables like packets in the network model. We also introduce a hidden variable H as a parent of the system calls variables to allow correlations among them. This hidden variable is designed to model the internal state of the machine, although such a semantic meaning is not imposed by our method. Put together, our system call model looks like Figure 7.1.

If the size of state space of the hidden variable H is m, the transition rate matrix of H is

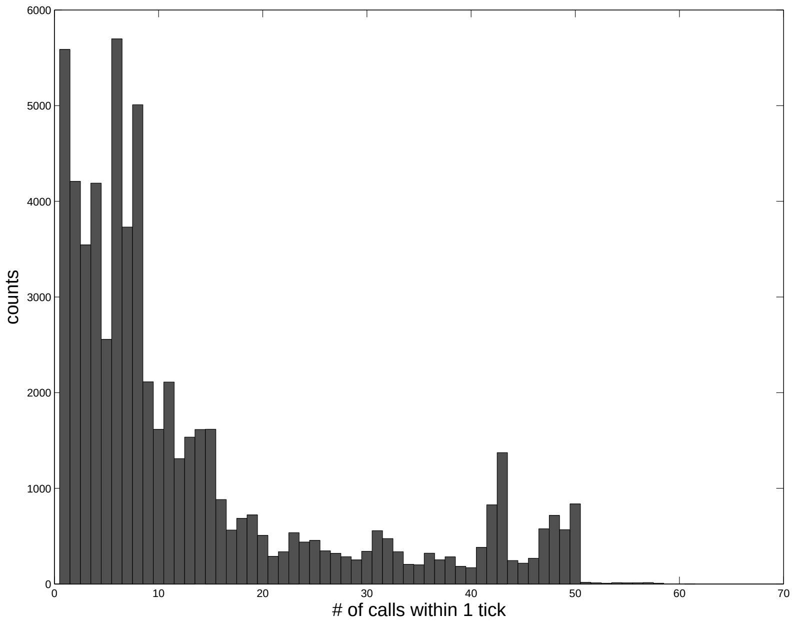


Figure 7.2: Histogram of the number of system calls within a tick

$$\mathbf{Q_H} = \begin{bmatrix} -q_{h_1} & q_{h_1h_2} & \cdots & q_{h_1h_m} \\ q_{h_2h_1} & -q_{h_2} & \cdots & q_{h_2h_m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{h_mh_1} & q_{h_mh_2} & \cdots & -q_{h_m} \end{bmatrix}.$$

And the transition intensity rate of the toggle variable $s \in S$ given the current value of its parent H is $q_{s|h_i}, i = 1, ..., m$.

To estimate the CTBN model parameters, we again use the expectation maximization (EM) algorithm. The expected sufficient statistics we need to calculate for our model are

- $\bar{M}_{h_ih_j}$, the expected number of times H transits from state i to j;

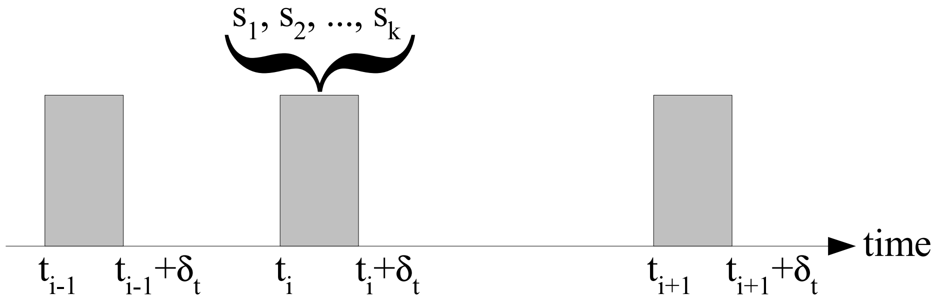


Figure 7.3: System call traces with a finite resolution clock (resolution = δ_t)

- $\bar{T}_{h_i}$, the expected amount of time H stays in state i; and

- $\bar{M}_{s|h_i}$, the expected number of times system call s is evoked when H is in state i.

The maximum likelihood parameters are

$$q_{h_i h_j} = \frac{\bar{M}_{h_i h_j}}{\bar{T}_{h_i}}$$
$$q_{s|h_i} = \frac{\bar{M}_{s|h_i}}{\bar{T}_{h_i}} \quad .$$

7.2 Parameter Estimation with Finite Resolution Clocks

Because of the finite resolution of computer clocks, multiple instantaneous events (system calls) occur within a single clock tick. Figure 7.2 shows the histogram of the number of system calls within a tick. Therefore in the audit logs, a batch of system calls may be recorded as being executed at a same time point, rather than their real time stamp, as a result of this finite time accuracy. However, the correct order of the events is kept in the logs. That is, we know exactly that system call S_2 follows S_1 if they are recorded in this order in the audit logs.

Thus all the system call timings are only partially observed. This type of partial observation has not previously been considered in CTBN inference. A typical trajectory τ over $[0, T]$ of system call data is shown in Figure 7.3: a batch of system calls are evoked at some time after t_i but before the next clock tick, followed by a quiet period of arbitrary length, and yet another bunch of events at some time after t_{i+1} and so on.

Let $\tau_{t1:t2}$ denote the evidence over interval $[t1, t2)$, $\tau_{t1:t2^+}$ denote the evidence over $[t1, t2]$, and $\tau_{t1^-:t2}$ denote the evidence over $(t1, t2)$. We define the vectors

$$\alpha_{t_i}^{-} = p(H_{t_i^-}, \tau_{0:t_i})$$

$$\beta_{t_i}^{+} = p(\tau_{t_i^+:T} | H_{t_i^+})$$

where $H_{t_i^-}$ is the value of H just prior to the transition at t_i, and $H_{t_i^+}$ is value just afterward. We also define the vectors

$$\alpha_{t_i} = p(H_{t_i}, \tau_{0:t_i^+})$$

$$\beta_{t_i} = p(\tau_{t_i:T} | H_{t_i})$$

where the evidence at the transition time t_i is included. We follow the forward-backward algorithm to compute α_{t_i} and β_{t_i} for all t_i at which there is an event. To do this, we split any interval $[t_i, t_{i+1})$ into a "spike" period $[t_i, t_i + \delta_t)$ (δ_t is one resolution clock), during which there is a batch of system calls, and a "quite" period $[t_i + \delta_t, t_{i+1})$ over which no events exist, and do the propagations separately.

For a "spike" period $[t_i, t_i + \delta_t)$, if the observed event sequence is $s_1, s_2, ..., s_k$, we construct an artificial Markov process X with the following intensity matrix.

$$\mathbf{Q_X} = \begin{bmatrix} \hat{\mathbf{Q}}_\mathbf{H} & \mathbf{Q_1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{Q}}_\mathbf{H} & \mathbf{Q_2} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \hat{\mathbf{Q}}_\mathbf{H} & \mathbf{Q_k} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \hat{\mathbf{Q}}_\mathbf{H} \end{bmatrix}.$$

where

$$\hat{\mathbf{Q}}_\mathbf{H} = \begin{bmatrix} -q_{h_1} - \sum_{s \in S} q_{s|h_1} & q_{h_1 h_2} & \dots & q_{h_1 h_m} \\ q_{h_2 h_1} & -q_{h_2} - \sum_{s \in S} q_{s|h_2} & \dots & q_{h_2 h_m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{h_m h_1} & q_{h_m h_2} & \dots & -q_{h_m} - \sum_{s \in S} q_{s|h_m} \end{bmatrix}$$

and

$$\mathbf{Q_i} = \begin{bmatrix} q_{s_i|h_1} & 0 & \dots & 0 \\ 0 & q_{s_i|h_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & q_{s_i|h_m} \end{bmatrix}$$

X tracks the evidence sequence $s_1 \rightarrow s_2 \rightarrow ... \rightarrow s_k$. $\mathbf{Q_X}$ is a square block matrix of dimension $m \times (k+1)$. Each block is an $m \times m$ matrix. The subsystem X has $k+1$ blocks of states. The first block represents the state of H before any events. The second block represents H after exactly one event, s_1, happens. The third block represents H after s_1 followed by s_2 happens, and so on. The last block represents H after all the events finish executing in order. The subsystem has zero transition intensities everywhere except along the sequence pass. The diagonal of $\hat{\mathbf{Q}}_{\mathbf{H}}$ is the same matrix as that of $\mathbf{Q}_{\mathbf{H}}$ except that the transition intensities of all the system call variables are subtracted. This is because the full system includes transitions that were not observed. While those transition rates were set to zero (to force the system to agree with the evidence), such conditioning does not change the diagonal elements of the rate matrix (Nodelman et al., 2002). Within each of the $k+1$ states of a block, H can freely change its value. Therefore, the non-diagonal elements of $\hat{\mathbf{Q}}_{\mathbf{H}}$ have the same intensities as $\mathbf{Q}_{\mathbf{H}}$. Upon transitioning, X can only transit from some state to another according to the event sequence. Therefore, most of the blocks are 0 matrices except those to the immediate right of the diagonal blocks. The transition behavior is described by the matrix Q_i. Q_i has 0 intensities on non-diagonal entries because H and S can not change simultaneously. The diagonal element $Q_i(h,h)$ is the intensities of event s_i happening, given the current value of the hidden state is h.

We take the forward pass as an example to describe the propagation; the backward pass can be performed similarly. Right before t_i, $\alpha_{t_i}^-$ has m dimensions. We expand it to $m(k+1)$ dimensions to form α_{t_i} which only has non-zero probabilities in the first m states. α_{t_i} now describes the distribution over the subsystem X. $\alpha_{t_i} e^{\mathbf{Q}_{\mathbf{X}} \delta_{\mathbf{t}}}$ represents the probability distribution at time $t_i + \delta_t$, given that some prefix of the observed sequence occurred. We take only the last m state probabilities to condition on the entire sequence happening, thus resulting in an m-dimensional vector, $\alpha_{t_i+\delta_t}$.

For a "quiet" period $[t_i+\delta_t, t_{i+1})$, no evidence is observed. Therefore $\alpha_{t_i+\delta_t}$ is propagated to $\alpha_{t_{i+1}}$ using $\hat{\mathbf{Q}}_{\mathbf{H}}$, the rate matrix conditioned on only H events occuring:

$$\alpha_{t_{i+1}} = \alpha_{t_i+\delta_t} \exp(\hat{\mathbf{Q}}_{\mathbf{H}}(\mathbf{t_{i+1}} - \mathbf{t_i} - \delta_{\mathbf{t}})) \ .$$

When we are done with the full forward-backward pass over the whole trajectory, we can calculate the expected sufficient statistics $\bar{M}_{h_ih_j}$, $\bar{T}_{h_i}$ and $\bar{M}_{s|h_i}$. Again, we refer to 3 for the algorithm.

7.3 Testing using Likelihood

Once we have learned the model from the normal process in the system call logs, we calculate the log-likelihood of a future process under the model. The log-likelihood is then compared to a predefined threshold. If it is below the threshold, a possible anomaly is indicated. With only a single hidden variable, these calculations can be done exactly.

7.4 Evaluation

In this section, we present our experiment results on HIDS.

7.4.1 Dataset

The dataset we used is the 1998 DARPA Intrusion Detection Evaluation Data Set from MIT Lincoln Laboratory. Seven weeks of training data that contain labeled network-based attacks in the midst of normal background data are publicly available at the DARPA website. The Solaris Basic Security Module (BSM) praudit audit data on system call logs are provided for research analysis. We follow Kang et al. (2005) to cross-index the BSM logs and produce a labeled list file that labels individual processes. The resulting statistics are shown on the table of Figure 7.4. The frequency of all the system calls appearing in the dataset is summarized in descending order on the table of Figure 7.5.

Week	# normal processes	# attack processes
1	786	2
2	645	4
3	775	20
4	615	331
5	795	10
6	769	24
7	584	0

Figure 7.4: DARPA BSM process summary.

7.4.2 Anomaly Detection

Our experimental goal is to detect anomalous processes. We train our CTBN model on normal processes only and test on a mixture of both normal and attack processes. The state space of the hidden variable H is set to 2. The log-likelihood of a whole process under the

System Call	# occurrence	System Call	# occurrence
close	123403	execve	1741
ioctl	68849	chdir	1526
mmap	60886	chroot	328
open	42479	unlink	26
fcntl	7416	chown	23
stat	6429	mkdir	4
access	2791	chmod	1

Figure 7.5: DARPA BSM system call summary

learned model represents the score of this process. We compare to the score with a predefined threshold to classify the process as a normal one or a system abuse.

We implement sequence time-delaying embedding (stide) and stide with frequency threshold (t-stide) for comparison (Warrender et al., 1999). These two algorithms build a database of all previously seen normal sequences of system calls and compare the testing sequences with it. They are straightforward and perform very well empirically on most of the system call log datasets. We choose the parameter k, the sequence length to be 5, and h, the locality frame length, to be 50. The results for t-stide are not shown in the following resulting graphs since they overlapped with stide in almost all cases.

Other approaches we compare against are nearest neighbor and one-class SVM with spectrum string kernel and edit distance kernel. We follow Hu et al. (2003) and transform a process into a feature vector, consisting of the occurrence numbers of each system call in the process. The nearest distance between a testing process and the training set of processes is assigned as the score. For one-class SVM, processes are composed of strings of system calls. Normal processes are used for learning the bounding surface and the signed distance to it is

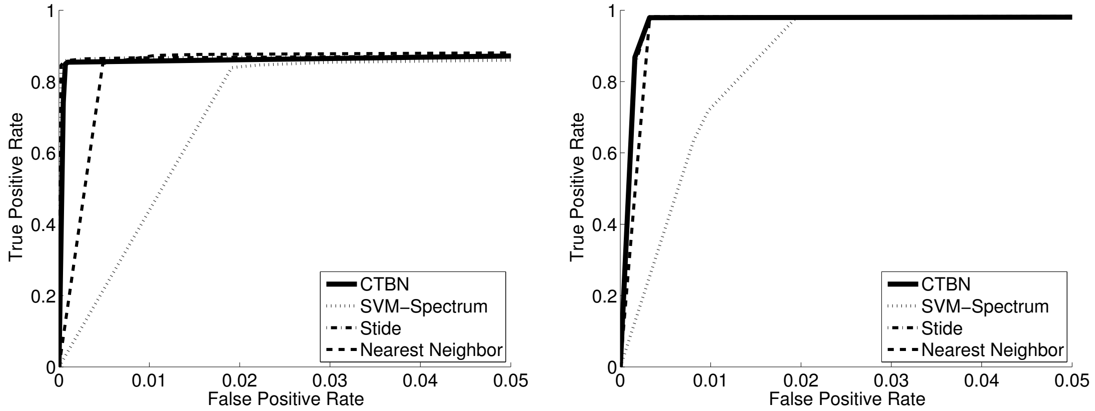


Figure 7.6: ROC curves for BSM data detection. Left: Training on week 1 and combined testing results on week 2 to 7; Right: Training on week 3 and test on week 4, Stide curve and CTBN curve overlap

assigned as the score. We set the sub-sequence length to be 5 and the parameter ν to be 0.5. As in Chapter 5, since the edit distance kernel results are dominated by the spectrum kernel, we do not show them.

Figure 7.6 displays the results from two experimental settings. In the left graph, we train the model on the normal processes from week 1 and test it on all the processes from weeks 2 to 7. In the right graph, we train on normal processes from week 3 and test it on all the processes from week 4, the richest in attack processes volume. Because attacks are relatively rare compared to normal traffic, we are most interested in the region of the ROC curves with small false positive rates. So we only show the curves in the area where the false positive rate falls in the region [0, 0.05]. Our CTBN method beats nearest neighbor and SVM with spectrum kernel in both experiments. stide performs slightly better than our method in the combined test, but achieves the same accuracy in the experiment using only week 3 and testing on week 4.

We conclude that by using a CTBN model, the dynamic behavior of the system calls on a computer is well described. The special feature of the data — finite resolution clocks — is successfully handled by the model. Our method is a powerful way to detect intrusions for HIDS.

Chapter 8

Conclusions

In the realm of temporal reasoning, we have introduced three additions to the CTBN literature. First, we added to CTBNs toggle variables, which describe instantaneous events without intrinsic states. We also derived parameter learning algorithms for them. Second, we demonstrated a Rao-Blackwellized particle filter with continuous evidence. We used a particle filter to sample a portion of the variables, and analytically integrated out the rest. Therefore, we decomposed the structure efficiently and reduced the sampling space. Finally, we demonstrated that we can learn and reason about data that contains imprecise timings, while still refraining from discretizing time. We introduced artificial variables to describe the dynamics of the sequential events under finite resolutions.

In the realm of intrusion detection, we have demonstrated a framework that performs well on two related tasks with very different data types. By concentrating purely on event timing, without the consideration of complex features, we were able to out-perform existing methods on real data. The continuous-time nature of our model aided greatly in modeling the bursty event sequences that occur in systems logs and network traffic. We did not have to resort to time slicing, either producing rapid slices that are inefficient for quite periods, or lengthy slices that miss the timing of bursty events. We do not require labeled data, thus vastly improving the automation of the detection.

A combination of the two sources of information (system calls and network events) would be straight-forward with the model we have produced. We believe it would result in more accurate detection. The collection of such data is difficult, however; we leave it as an interesting next step.

Bibliography

Agosta, J. M., Chandrashekar, J., Giroire, F., Livadas, C., and Xu, J. (2007a). Approaches to anomaly detection using host network-traffic traces. In *Neural Information Processing Systems Workshop on Machine Learning for Systems Problems(MLSys)*.

Agosta, J. M., Duik-Wasser, C., Chandrashekar, J., and Livadas, C. (2007b). An adaptive anomaly detector for worm detection. In *Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*.

A.Hofmeya, S., Forrest, S., and Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180.

Asmussen, S., Nerman, O., and Olsson, M. (1996). Fitting phase-type distributions via the EM algorithm. *Scandavian Journal of Statistics*, 23:419–441.

Beirlant, J., Dudewicz, E., Gyöfi, L., and van der Meulen, E. C. (1997). Nonparametric entrop estimation: An overview. *International Journal of Mathematical and Statistical Sciences*, 6(1):17–39.

Cha, B. (2005). Host anomaly detection performance analysis based on system call of neuro-fuzzy using soundex algorithm and n-gram technique. In *Systems Communications (ICW)*.

Cohn, I., El-Hay, T., Friedman, N., and Kupferman, R. (2009). Mean field variational approximation for continous-time bayesian networks. In *Uncertainty in Artificial Intelligence*.

Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38.

Dewaele, G., Fukuda, K., and Borgnat, P. (2007). Extracting hidden anomalies using sketch and non Gaussian multiresulotion statistical detection procedures. In *ACM SIGCOMM*.

Doucet, A., de Freitas, N., Murphy, K., and Russel, S. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *Uncertainty in Artificial Intelligence*.

El-Hay, T., Friedman, N., and Kupferman, R. (2008). Gibbs sampling in factorized continous-time markov processes. In *Uncertainty in Artificial Intelligence*.

Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In *International Conference on Machine Learning*.

Eskin, E., Arnold, A., Prerau, M., Portnoy, L., and Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In Barbara, D. and Jajodia, S., editors, *Applications of Data Mining in Computer Security*. Kluwer.

Fan, Y. and Shelton, C. R. (2008). Sampling for approximate inference in continuous time bayesian networks. In *Symposium on Artificial Intelligence and Mathematics*.

Forrest, S., A.Hofmeya, S., Somayaji, A., and A.Longstaff, T. (1996). A sense of self for unix processes.

Gopalratnam, K., Kautz, H., and Weld, D. S. (2005). Extending continuous time Bayesian networks. In *National Conference on Artificial Intelligence*, pages 981–986.

Hu, W., Liao, Y., and Vemuri, V. (2003). Robust support vector machines for anomaly detection in computer security. In *International Conference on Machine Learning and Applications*.

Kang, D.-K., Fuller, D., and Honavar, V. (2005). Learning classifiers for misuse detetction using a bag of system calls representation. In *IEEE International Conferences on Intelligence and Security Informatics*.

Karagiannis, T., Papagiannaki, K., and Faloutsos, M. (2005). BLINC: Multilevel traffic classification in the dark. In *ACM SIGCOMM*.

Khan, Z., Balch, T., and Dellaert, F. (2004). A rao-blackwellized particle filter for eigentracking. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Kruegel, C., Mutz, D., Robertson, W., and Valeur, F. (2003). Bayesian event classification for intrusion detection. In *Annual Computer Security Applications Conference*.

Lakhina, A., Crovella, M., and Diot, C. (2005). Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, pages 21–26.

Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., and Srivastava, J. (2003). A compare study of anomaly detection schemes in network intrusion detection. In *SIAM International Conference on Data Mining*.

Lee, W. and Stolfo, S. J. (1998). Data mining approaches for intrusion detection. In *USENIX Security Symposium*.

Leslie, C., Eskin, E., and Noble, W. S. (2002). The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing 7:566-575*.

Neuts, M. F. (1975). Probability distributions of phase type. In *Liber Amicorum Prof. Emeritus H. Florin*, pages 173–206. Department of Mathematics, University of Louvain, Belgium.

Neuts, M. F. (1981). *Matrix-Geometric Solutions in Stochastic Models — An Algorithmic Approach*. John Hopkins University Press, Baltimore.

Ng, B., Pfeffer, A., and Dearden, R. (2005). Continuous time particle filtering. In *National Conference on Artificial Intelligence*, pages 1360–1365.

Nodelman, U., Koller, D., and Shelton, C. R. (2005a). Expectation propagation for continuous time Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 431–440.

Nodelman, U., Shelton, C. R., and Koller, D. (2002). Continuous time Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 378–387.

Nodelman, U., Shelton, C. R., and Koller, D. (2003). Learning continuous time Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 451–458.

Nodelman, U., Shelton, C. R., and Koller, D. (2005b). Expectation maximization and complex duration distributions for continuous time Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 421–430.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C*. Cambridge University Press, second edition.

Rieck, K. and Laskov, P. (2007). Language models for detection of unknown attacks in network traffic. In *Journal in Computer Virology*.

Saria, S., Nodelman, U., and Koller, D. (2007). Reasoning at the right time granularity. In *Uncertainty in Artificial Intelligence*.

Schulz, D., Fox, D., and Hightower, J. (2003). People tracking with anonymous and id-sensors using rao-blackwellised particle filters. In *International Joint Conference on Artificial Intelligence*.

Sim, R., Elinas, R., and Little, J. J. (2007). A study of the rao-blackwellised particle filter for efficient and accurate vision-based slam. *International Journal of Computer Vision*, (3):303–318.

Soule, A., Salamatian, K., and Taft, N. (2005). Combining filtering and statistical methods for anomaly detection. In *Internet Measurement Conference*, pages 331–344.

Soule, A., Salamatian, L., Taft, N., Emilion, R., and Papagiannali, K. (2004). Flow classification by histogram. In *ACM SIGMETRICS*.

Tandon, G. and Chan, P. K. (2005). Learning useful system call attributes for anomaly detection. In *The Florida Artificial Intelligence Research Society Conference, pp. 405-410*.

Warrender, C., Forrest, S., and Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy, IEEE Computer Society*.

Xu, J. and Shelton, C. R. (2008). Continuous time bayesian networks for host level network intrusion detection. In *European Conference on Machine Learning*.

Xu, K., Zhang, Z.-L., and Bhattacharyya, S. (2005). Profiling internet backbone traffic: Behavior models and applications. In *ACM SIGCOMM*.

Ye, N., Emran, S. M., Chen, Q., and Vilbert, S. (2002). Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions of Computers*, 51(7):810–820.

Yeung, D.-Y. and Chow, C. (2002). Parzen-window network intrusion detectors. In *International Conference on Pattern Recognition*.

Yeung, D.-Y. and Ding, Y. (2002). User profiling for intrusion detection using dynamic and static behavioral models. *Advances in Knowledge Discovery and Data Mining*, 2336:494–505.

Zuev, D. and Moore, A. (2005). Internet traffic classification using Bayesian analysis techniques. In *ACM SIGMETRICS*.