Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation

Yu Gu, Andrew McCallum, Don Towsley
Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003

Abstract

We develop a behavior-based anomaly detection method that detects network anomalies by checking the current network traffic against a baseline distribution estimated using the Maximum Entropy technique. By computing a measure related to the relative entropy of the network traffic under observation with respect to the baseline distribution, we are able to distinguish anomalies from benign traffic. The Maximum Entropy technique provides a flexible and fast approach to estimate the baseline distribution, which gives the network administrator a multi-dimensional view of the network traffic. Our method can detect anomalies that either cause abrupt changes in the network or increase malicious traffic slowly. It also provides information about the types of the anomalies detected. The method is easy to deploy and very scalable.

I. Introduction

Malicious abuses of the Internet are commonly seen in today's Internet traffic. Anomalies such as worms, port scans, denial of service attacks, etc. can be found at any time in the network traffic. These anomalies waste network resources, cause performance degradation of network devices and end hosts, and lead to security issues concerning all Internet users. Thus, accurately detecting such anomalies has become an important issue to the network community.

Much research has been devoted to the problem of anomaly detection in network traffic. Several metrics have been developed to measure and capture anomalies in the traffic. In anomaly based approaches, the network anomalies are characterized by rules extracted from previous anomalies. Traffic under observation is compared with all the rules, and an alarm is raised if a match is found. Abrupt changes in the network traffic has also been used as an indicator of network anomalies. If the statistics under observation show too much volatility, an alarm is raised.

In this paper, we develop a network anomalies detection technique which is based on a baseline distribution estimated using the Maximum Entropy estimation and a measure related to the relative entropy of the network traffic. We divide packets in the network traffic into classes along multiple dimensions and inspect the distribution

of these packet classes. Our approach exploits the idea of behavior-based anomaly detection. A *baseline distribution* of the packet classes in the benign traffic is determined by learning a density model from a set of pre-labeled training data. The empirical distribution of packet classes under observation is then compared to this baseline distribution. If the two distributions differ, we show that the most significant packet classes that cause the difference in the distributions contain packets related to an anomaly.

The relative entropy approach described in this work has many advantages. First, it gives the administrators a multi-dimensional view of the packets in the network traffic. The Maximum Entropy estimation framework provides a flexible and fast way to estimate and represent the baseline distribution based on the packet classes. Second, not only can it detect anomalies that cause abrupt changes in the network traffic, it can also detect anomalies that increase the traffic slowly. A large deviation from the baseline distribution can only be caused by packets that make up an unusual portion of the traffic. Thus if an anomaly occurs, no matter how slowly it increases its traffic, as long as its traffic exceeds a certain level in the overall network traffic, the relative entropy inevitably increases to a certain level that can be detected. Third, it provides information about the type of anomaly detected. Our approach identifies those packet classes that increase the relative entropy thus providing the network administrator information related to the types of the anomalies detected. Fourth, it is scalable as it only requires a constant amount of memory and consists solely of counting packets in the traffic, without requiring any per flow information. Finally, deploying the approach is also easy due to the simple computation needed to check for the anomalies.

Our approach divides into two phases. Phase one is to learn the baseline distribution and phase two is to detect anomalies in the observed traffic. In the first phase, we first divide packets into multi-dimensional packet classes according to the packets' protocol information and destination port numbers. These packet classes serve as the domain of the probability space. Then, the baseline distribution of the packet classes is determined by learning a density model from the training data using Maximum Entropy estimation. The training data is a pre-labeled data set with the anomalies labeled by human and packets labeled as anomalous removed. During the second phase, an observed network traffic trace is given as the input. The relative entropy of the packet classes in the observed traffic trace with respect to the baseline distribution is computed. The packet classes that contribute significantly to the relative entropy are then recorded. If certain packet classes keep contributing significantly to the relative entropy, anomaly warnings are generated and the corresponding packet classes are reported. This corresponding packet class information reveals the protocols and the destination port numbers related to the anomalies.

We test the approach over a set of real traffic traces. One of them is used as the training set and the others are used as the testing data sets. The experimental results show that our approach gives high positives with low false negatives and false positives. We also show that even with an unlabeled training set, the approach still gives good results.

The rest of the paper is organized as follows. In Section II, we review the related work in network anomaly detection and also that in the field of machine learning related to the Maximum Entropy framework used in this work. Section III describes how we divide packets in the traffic into different packet classes. In Section IV, we introduce the log-linear model which describes the distribution of different classes of packets, and describe how it is built from the training data using the Maximum Entropy technique. In Section V, we describe a method based on relative entropy to detect anomalies in the network traffic. Section VI gives details of our experimentation, including the data collection, model training results, the experimental results and comparison with other detection methods. In Section VII, we discuss several aspects of the relative entropy method in anomaly detection, including some future work. The last section summarizes the whole paper.

II. Related work

A variety of tools have been developed for the purpose of network traffic anomaly detection. Some detect anomalies by comparing the traffic pattern or the packets with a set of predefined rules which describe characteristics of the anomalies. These are called anomaly based detection. Many of the rules or policies used in Snort [11] and Bro [9] fall in this category. The time cost for these approaches is proportional to the size of the set of the anomaly rules defined, which affects the scalability of these approaches. Furthermore they are not sensitive to anomalies that have not been previously defined. Our work is a behavior based approach and requires little computation.

A number of the existing approaches are variations on the change detection method. In [2], Brutlag uses the Holt Winter forecasting model to capture the history of network traffic variations and to predict future traffic rate. The prediction of the future traffic rate is given in the form of a confidence band, which yields upper and lower bounds on the predicted traffic rate. When the variance of the network traffic continues to fall outside of the confidence band, an alarm is raised. In [1], Barford *et al*. use wavelet analysis to remove from the traffic the predictable ambient part and then study the variations in the network traffic rate. Network anomalies are detected by applying a threshold to a deviation score computed from the analysis. In [13], Thottan and Ji take management information base (MIB) data collected from routers as time series data and use an auto-regressive process to model the process. Network anomalies are detected by inspecting abrupt changes in the statistics of the data. In [14], Wang *et al*. take the difference in the number of SYNs and FINs (RSTs) collected within one sampling period as time series data and use a non-parametric Cumulative Sum (CUSUM) method to detect SYN flooding by detecting the change point of the time series. While these methods can detect anomalies that cause unpredicted changes in the network traffic, they may be deceived by attacks that increase their traffic slowly. If the increase in the anomaly traffic is slow enough, the corresponding statistics may change so smoothly that they can evade the change point detection mechanism. Our work can detect anomalies regardless of how slowly the traffic is increased and report on the types of anomalies detected.

There are also works on anomaly detection that use relative entropy or other information theoretic approaches. In [6], Lee and Xiang study several information theoretic measures for anomaly detection. They show that information theoretic measures can be used to help select appropriate anomaly detection models and explain the performance of the models. In their study on the network traffic, entropy and conditional entropy are used to help data partitioning and model parameter setting. In [12], Staniford *et al*. use information theoretic measures to help detect stealthy port scans. Probability tables of feature instances and multi-dimensional tables of conditional probabilities observed are maintained and are used to count the number of features in observed events. Then, when a packet is observed, the anomalousness of the packet is evaluated using the negative log likelihood of this packet with respect to its features. Our work in this paper exploits relative entropy directly as a metric on the anomalousness in the traffic. We apply a systematic framework, Maximum Entropy Estimation, to estimate the baseline distribution of different types of packets present in the network traffic, and our approach is not limited to locating port scans.

Maximum Entropy is a general machine learning technique, which has been widely used in the fields of machine learning, information retrieval, computer vision, and econometrics, etc. The Maximum Entropy estimation technique in this work was introduced in [10], where Pietra *et al*. introduce a systematic way to induce features from random fields using the Maximum Entropy technique. In [8], McCallum builds, on [10], an efficient approach to induce features of Conditional Random Fields (CRFs). CRFs are undirected graphical models (also known as random fields) used to calculate the conditional probability of values on designated output nodes given values assigned to other designated input nodes. And in [7], Malouf gives a detailed comparison of several Maximum Entropy parameter estimation algorithms. In our work, we use the L-BFGS algorithm implemented by Malouf to estimate the parameters in the Maximum Entropy model.

III. Packet classification

In this section, we describe how we divide packets in the network traffic into a set of packet classes. This set of packet classes is the common domain of the probability densities used in our work.

Our work deals with TCP and UDP packets only. TCP and UDP are the two most popular protocols used in today's Internet. They carry the majority of today's network traffic and can be easily abused. For this reason, this work focuses on anomalies concerning TCP and UDP packets. We consider the protocol information carried by TCP and UDP packets and the destination port numbers in the packets as two of the most important packet head fields in the network traffic. In order to study the distribution of these packets, we divide the packets into a set of two-dimensional classes according to the protocol information and the destination port number in the packet header.

In the first dimension, packets are divided into four classes according to the protocol related information. First, packets are divided into the classes of TCP and UDP packets. Two other classes are further split from the TCP

packet class according to whether or not the packets are SYN and RST packets. A SYN packet is the first packet to establish a TCP connection. It causes the receiving host to allocate resources for the coming TCP connection. Thus, SYN packets are often used to perform SYN flooding attacks or port scans. RST packets are generated when a TCP connection has to be reset or rejected. For example, in the case of an attempt to connect to a nonexisting port, a TCP RST packet is sent from the target host. So an increase in TCP RST packets reveals an increase in TCP connection failures, which may relate to network anomalies.

In the second dimension, packets are divided into 587 classes according to their destination port numbers. Port numbers often determine the services related to the packet exchange. Web services, for example, which generate the dominant traffic in the Internet, often have port number 80; FTP services often have port numbers 22 and 23; and mail services often have port number 25. According to the Internet Assigned Numbers Authority [5], port numbers are divided into three categories: *Well Known Ports*, *Registered Ports*, and *Dynamic and/or Private Ports*. *Well Known Ports* are those from 0 through 1023, *Registered Ports* are those from 1024 through 49151, and *Dynamic and/or Private Ports* are those from 49152 through 65535. In order to reduce the total number of packet classes and make classification possible, packets with different destination port numbers are grouped into different classes according to whether they are 'well known ports', 'registered ports', or 'dynamic and/or private ports'. Packets with destination port numbers between 0 and 1023 are divided into classes of 10 port numbers each. Since packets with port number 80 comprise the majority of the network traffic, they are placed into a single class. Consequently, packets with destination port numbers in the range $[81, 89]$ form another separate class. Furthermore, packets with destination port numbers in the range $[1020, 1023]$ form another class. This produces 104 packet classes. Packets with destination port numbers between 1024 and 49151 are divided into 482 additional classes, with each class covering 100 port numbers with the exception of the class that covers port numbers from 49124 to 49151, which contains 28 port numbers. Packets with destination port numbers larger than 49151 are grouped into a single class. Thus, in this dimension, packets are divided into a total of $104 + 482 + 1 = 587$ classes.

Altogether, the set of the two-dimensional classes consists of $4 * 587 = 2348$ packet classes. These packet classes form the common domain of the probability spaces in this paper. Our work is based on studying the distribution of these packet classes in the network traffic. We estimate the distribution of different packets in benign traffic according to this classification, and use it as the baseline distribution to detect network traffic anomalies.

Note that this packet classification does not consider in the different distributions of packet classes at different times of the day. Thus multiple such distributions should be built for different times. However, time factors could be added as a dimension in the packet classes. A possible solution is given in the subsection VII-C, discussing the future work.

IV. Maximum Entropy estimation of the packet classes distribution

Maximum Entropy estimation is a framework for estimating a parametric probability distribution model from the training data. Suppose the distribution model is required to satisfy a set of constraints reflecting properties of the training data, the Maximum Entropy estimation then follows the principle that a good model of the data should be one that satisfies these constraints only and makes no other assumptions. Here, making no other assumptions implies requiring maximum uniformity of the model. In other words, we should represent the data by the most uniform model of all the models that satisfy the constraints. A mathematical measurement of the uniformity of a distribution P is its entropy:

$$-\sum_{\omega\in\Omega} P(\omega)\log P(\omega). \tag{1}$$

This entropy is bounded from below by 0 corresponding to the case that there is no uncertainty, and from above by the log of the cardinality of Ω corresponding to uniformity.

Let Ω be the set of packet classes defined in the previous section. Given a sequence of packets $\mathcal{S} = \{x_1, \ldots, x_n\}$ as the training data, the empirical distribution $\tilde{P}$ over Ω in this training data is

$$\tilde{P}(\omega) = \frac{\sum \mathbf{1}(x_i \in \omega)}{n}, \tag{2}$$

where $\mathbf{1}(X)$ is an indicator function that takes value 1 only if X is true and 0 otherwise.

Suppose we are given a set of feature functions $\mathcal{F} = \{f_i\}$, f_i is a feature function, which is an indicator function $f_i : \Omega \mapsto \{0, 1\}$[1]. By using the Maximum Entropy estimation, we are looking for a density model P that satisfies $E_P(f_i) = E_{\tilde{P}}(f_i)$ for all $f_i \in \mathcal{F}$ and has the maximum entropy. In [10], it has been proved that under the constraints that $E_P(f_i) = E_{\tilde{P}}(f_i)$ for all $f_i \in \mathcal{F}$, the Maximum Entropy estimate is guaranteed to be (a) unique, and (b) the same as the maximum likelihood estimate using the generalized Gibbs distribution, which is in a log-linear form of

$$P(\omega) = \frac{1}{Z}\exp(\sum_i \lambda_i f_i(\omega)) \tag{3}$$

For each feature f_i, a parameter $\lambda_i \in \Lambda$ determines its weight in the model, Λ is the set of parameters for the feature functions in the model. Z is a normalization constant that ensures that the sum of the probabilities over Ω is 1. Maximizing the likelihood of the distribution in the form of (3) with respect to $\tilde{P}$ is equivalent to minimizing the Kullback-Leibler divergences of $\tilde{P}$ with respect to P

$$P = \arg\min_P D(\tilde{P}\|P)$$

[1]In general, the range of a feature function can be the real numbers. In this work, we use only indicator functions as feature functions

as

$$\prod_{\omega\in\Omega} P(\omega)^{\sum \mathbf{1}(x_i\in\omega)} \propto \exp(-D(\tilde{P}\|P)).$$

Feature functions in the log-linear model (3) represent certain properties of the training data in the learned model. For example, the most conspicuous feature of the current Internet traffic is that it is most commonly comprised of TCP packets. The log-linear model can reflect this observation by using a single feature function $f(\omega) = \mathbf{1}(\omega \text{ consists of TCP packets})$, and assigning 2.246 to the parameter λ_f which determines the weight of f

$$P(\omega) = \frac{1}{Z}\exp(2.246\mathbf{1}(\omega \text{ consists of TCP packets}))$$

This means that a TCP packet is about $9.45 \sim \exp(2.246)$ times more likely than a non-TCP packet. In general, a feature function can be any indicator function that maps Ω to $\{0, 1\}$. For the sake of efficiency, feature functions are often selected to express the most important characteristics of the training data in the learned log-linear model, and in return, the log-linear model expresses the empirical distribution with the fewest feature functions and parameters.

Thus, the Maximum Entropy estimation procedure consists of two parts: feature selection and parameter estimation. The feature selection part selects the most important features of the log-linear model, and the parameter estimation part assigns a proper weight to each of the feature functions. These two parts are performed iteratively such that the set of feature functions increases one at a time and the weights for the feature functions in the log-linear model are adjusted as each new feature function is added. As the feature functions in the model are added, the model approaches to the empirical distribution; and the parameter estimation ensures that the model satisfies the Maximum Entropy principle. In the following, we describe each part in turn. More details can be found in [10].

A. Feature selection

The feature selection step is a greedy algorithm which chooses the best feature function from a set of candidate feature functions that minimizes the difference between the model distribution and the empirical distribution.

Let Ω be the set of all packet classes, $\tilde{P}$ the empirical distribution of Ω in the training data, and $\mathcal{F}$ a set of candidate feature functions. At the initial state, no information is known about the training data. So the initial model distribution over Ω is

$$P_0(\omega) = \frac{1}{Z}, \qquad Z = |\Omega|,$$

which is a uniform distribution over Ω. The subscript 0 indicates that there is no feature in the model.

Suppose g is a feature function $g \in \mathcal{F}$. We define the 1-parameter family of distributions $\{P_{\lambda_g,g} : \lambda_g \in \mathcal{R}\}$ as follows:

$$P_{0,\lambda_g,g}(\omega) = \frac{1}{Z'}\exp(\lambda_g g(\omega)) \tag{4}$$

This family of distribution is referred to as the induction of P_0 by g. We also define

$$G_{P_0}(\lambda_g, g) = D(\tilde{P}||P_0) - D(\tilde{P}||P_{0,\lambda_g,g}). \tag{5}$$

Here, $G_{P_0}(\lambda_g, g)$ is the difference of the two Kullback-Leibler divergences related to P_0 and $P_{0,\lambda_g,g}$. It is the improvement that feature g brings to the model with its weight λ_g. We define $G_{P_0}(g)$ to be the greatest improvement by adding g to P_0:

$$G_{P_0}(g) = \sup_{\lambda_g} G_{P_0}(\lambda_g, g) \tag{6}$$

and $G_{P_0}(g)$ is referred to as the gain of the candidate feature g. In order to minimize the difference between the model distribution and the empirical distribution, the feature function in $\mathcal{F}$ with the maximum gain is chosen. We name the selected feature function f_1

$$f_1 = \arg\max_g G_{P_0}(g) \tag{7}$$

and obtain the new model with one feature function selected as

$$P_1(\omega) = \frac{1}{Z}\exp(\lambda_1 f_1(\omega)). \tag{8}$$

Now let P_i be a model with i feature functions selected

$$P_i(\omega) = \frac{1}{Z}\exp(\sum_{j=1}^{i}\lambda_j f_j(\omega)). \tag{9}$$

and we want to select the $i+1^{st}$ feature function. Similar to the above procedure, let g be a feature function in $\mathcal{F}\backslash\{f_1, \dots f_i\}$ that has not yet been selected. We obtain

$$P_{i,\lambda_g,g}(\omega) = \frac{1}{Z'}\exp(\sum_i \lambda_i f_i(\omega))\exp(\lambda_g g) \tag{10}$$

Adding a new feature function would require all the parameters to be adjusted, which makes it computationally expensive to compute the exact form of the model with g added for all g in the candidate feature function set. As a compromise, the feature selection is based upon an approximation. The improvement of adding a new feature g is approximated by adjusting only the weight of the parameter of g and keeping the remaining parameters unchanged. Considering the large number of candidate feature functions, this approximation makes feature selection practical.

With this approximation, we have

$$\begin{aligned} & G_{P_i}(\lambda_g, g) \\ = \quad & D(\tilde{P}||P_i) - D(\tilde{P}||P_{i,\lambda_g,g}) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{\omega} \tilde{P}(\omega) \log \frac{P_{i,\lambda_g,g}}{P_i(\omega)} \\
&= \lambda_g E_{\tilde{P}}(g) - \log E_{P_i}(\exp(\lambda_g g)),
\end{aligned}
$$

which is a concave function with respect to λ_g, and

$$G_{P_i}(g) = \sup_{\lambda_g} G_{P_i}(\lambda_g, g). \tag{11}$$

The feature function g with the largest gain $G_{P_i}(g)$ is selected as the $i+1^{st}$ feature function to the model.

It is also shown in [10] that, for a candidate feature function g in the form of an indicator function, $G_P(\lambda_g, g)$ is maximized by

$$\hat{\lambda}_g = \arg\max_{\lambda_g} G_P(\lambda_g, g) = \log \left\{ \frac{E_{\tilde{P}}(g)(1 - E_P(g))}{E_P(g)(1 - E_{\tilde{P}}(g))} \right\} \tag{12}$$

in which $E_{\tilde{P}}(g)$ is the expected value of g with respect to the distribution of $\tilde{P}$ and $E_P(g)$ is the expected value of g with respect to the distribution of P. At this value of λ_g,

$$G_P(g) = G_P(\hat{\lambda_g}, g) = D(B_{\tilde{P}} || B_P) \tag{13}$$

where $B_{\tilde{P}}$ and B_P are random variables of Bernoulli distributions given by

$$
\begin{aligned}
P(B_{\tilde{P}} = 1) = E_{\tilde{P}}(g) \qquad & P(B_{\tilde{P}} = 0) = 1 - E_{\tilde{P}}(g) \\
P(B_P = 1) = E_P(g) \qquad & P(B_P = 0) = 1 - E_P(g).
\end{aligned}
$$

After a new feature function is added to the log-linear model, the parameter estimation procedure will update the weights of all feature functions in the model to minimize the Kullback-Leibler divergence between the model distribution and the empirical distribution.

B. Parameter estimation

Given a set of training data and a set of selected feature functions $\{f_i\}$, the set of parameters is then estimated. The Maximum Entropy estimation locates a set of parameters $\Lambda = \{\lambda_i\}$ in (3) for $\{f_i\}$ that minimizes the Kullback-Leibler divergence of $\tilde{P}$ with respect to P:

$$\Lambda = \arg\min_{\Lambda} \sum_{\omega \in \Omega} \tilde{P}(\omega) \log \frac{\tilde{P}(\omega)}{P(\omega)}. \tag{14}$$

Computing these parameters in the Maximum Entropy model directly is usually not easy. There are a number of numerical methods that can be exploited, including the *Generalized Iterative Scaling* method [3], *Improved Iterative Scaling* method [10], as well as general purpose optimization techniques such as gradient ascent, conjugate gradient, and variable metric methods [7]. In our work, we use the L-BFGS Maximum Entropy estimation algorithm

"tao_lmvm" implemented by Malouf in [7]. By using as inputs the empirical distribution, the set of features $\{f_i\}$, and the values of the features under each packet class, the Maximum Entropy estimator can compute the set of parameters $\{\lambda_i\}$ that minimizes the Kullback-Leibler divergence of the empirical distribution and the distribution defined by the log-linear model.

C. Model induction

The log-linear density model is built by iterating the above two steps. The algorithm begins with a uniform distribution over all packet classes. Then in the feature selection steps, new feature functions are added to the model as described in Section IV-A, and in the parameter estimation steps, the weights for the feature functions are adjusted as in Section IV-B. This procedure is iterated until some stopping criterion is met. This stopping criterion could be either that the Kullback-Leibler divergence of P with respect to $\tilde{P}$ is less than some threshold value, or that the gain of adding a new feature function is too small to improve the model.

The feature functions are selected from a set of candidate feature functions. Since the domain Ω in our work consists of packet classes different in the protocols and the destination port numbers, our candidate feature functions set comprises of three sets of indicator functions. The first set of indicator functions checks the packet's protocol information, the second set of indicator functions checks the packet's destination port number, and the third set checks both the packet's protocol information and the destination port number.

For $\omega \in \Omega$, the first set consists of four feature functions that check the protocol information of the packets in ω. Each feature function $f(\omega)$ would result in 1 if the packets in ω satisfy the corresponding protocol information and 0 otherwise. For example, the feature function that checks whether the packets in ω are TCP packets is

$$f_{TCP}(\omega) = \begin{cases} 1, & \text{packets in } \omega \text{ are TCP packets;} \\ 0, & \text{otherwise.} \end{cases}$$

The second set consists of 587 feature functions that check the destination port number of the packets in ω. For example, the feature function that check whether the packets in ω are targeted at ports between 20 and 29 is

$$f_{[20,29]}(\omega) = \begin{cases} 1, & \text{packets in } \omega \text{ targeted at ports} \\ & \text{between 20 and 29;} \\ 0, & \text{otherwise.} \end{cases}$$

And the third set consists of $4 * 587 = 2348$ feature functions that check both the protocol information and the destination port number of the packets in ω. For example, the feature function that checks whether the packets in

- Initial Data:
  A set of training data with empirical distribution $\tilde{P}$,
  a set of candidate feature functions $\mathcal{F}$,
  and an initial density model P_0, $P_0(\omega) = \frac{1}{Z}$, $Z = |\Omega|$
- Iterated steps:
  (0) Set $n = 0$
  (1) Feature selection
  For each feature function $g \in \mathcal{F}$, $g \notin \{f_i\}$, compute the gain $G_{P_n}(g)$
  Let f_{n+1} be the feature function with the largest gain
  (2) Parameter Estimation
  Update all the parameters and set P_{n+1} to be the updated model
  (3) Check the iteration stopping criterion
  If the iteration stopping criterion is not met, set $n = n + 1$, goto (1).
  Otherwise, return the learned model P_{n+1}.

Fig. 1. Model induction algorithm

ω are TCP SYN packets and the destination port number of the packets falls between 20 and 29 is

$$f_{SYN\times[20,29]}(\omega) = \begin{cases} 1, & \text{packets in } \omega \text{ are TCP SYN} \\ & \text{packets and are targeted} \\ & \text{at ports between 20 and 29;} \\ 0, & \text{otherwise.} \end{cases}$$

Figure 1 shows our model induction algorithm. Step (1) performs the feature selection procedure, and step (2) performs the parameter estimation procedure. As more and more feature functions are added to the model, the induction algorithm is able to reduce the Kullback-Leibler divergence between the empirical distribution $\tilde{P}$ and the model distribution P. In our work, this feature selection and parameter estimation process is iterated until the Kullback-Leibler divergence is less than some threshold value.

The training data used are pre-labeled by humans and the packets related to the labeled anomalies are not used in computing the empirical distribution $\tilde{P}$. In this way, we treat the packet classes distribution defined by the log-linear model in (3) from the Maximum Entropy estimation as the baseline distribution, and are now able to compute the relative entropy of any given network traffic.

V. Detecting network traffic anomalies using relative entropy

The relative entropy shows the difference between the distribution of the packet classes in the current network traffic and the baseline distribution. If this difference is too large, it indicates that a portion of some packet classes that rarely appear in the training data increases significantly in the current network traffic, or that appear regularly decrease significantly in the current network traffic. In other words, this serves as an indication of anomalies in the network traffic. In our work, we concern only the cases when anomaly traffic increases. In the following, we

describe a sliding window algorithm to detect network traffic anomalies based on this network traffic's relative entropy information.

We divide time into intervals of fixed length δ. Suppose the traffic in an interval contains the packet sequences $\{x_1, \ldots, x_n\}$, the empirical distribution $\tilde{P}$ of the packet classes in this interval is

$$\tilde{P}(\omega) = \frac{\sum \mathbf{1}(x_i \in \omega)}{n}, \tag{15}$$

For each class of packets, we define

$$D_{\tilde{P}\|P}(\omega) = \tilde{P}(\omega) \log \frac{\tilde{P}(\omega)}{P(\omega)}, \tag{16}$$

Here, P is the baseline distribution obtained from Maximum Entropy estimation. This gives us a quantitative value that describes the distortion of the distribution for each packet class ω from that of the baseline distribution. If $D_{\tilde{P}\|P}(\omega)$ is large, the portion of this class of packets deviates a lot from that in the baseline distribution, which is used as an indication of anomalies. Note the sum of $D_{\tilde{P}\|P}(\omega)$ over Ω is the relative entropy of $\tilde{P}$ with respect to P, and from now on we refer to $D_{\tilde{P}\|P}(\omega)$ as the 'relative entropy of class ω'.

We then use a three parameter 'sliding window' detection approach. We keep recording packet classes that have their divergences larger than a threshold d. If for a certain packet class ω, the number of times that $D_{\tilde{P}\|P}(\omega) > d$ exceeds h times in a time window w, an alarm is raised together with the packet class information ω, which reveals the corresponding protocol and port number.

VI. Experimental results

In this section, we present some experimental results. The experiments were carried on a set of seven hours' real network traffic trace. We use one hour trace as the training data and the rest six hours' trace as the test data. In the following, we first describe how the data are collected and labeled. Then we describe the training process and the log-linear density model as a result of the Maximum Entropy estimation algorithm. We then give some examples of network anomalies that show themselves with high relative entropy. We give the performance of the anomaly detection approach described in the previous section on these data collections, and compare it with the Holt-Winter algorithm.

A. Data collection and labeling

We test the approach using real network traffic data collected at the UMASS Internet gateway router. The UMASS campus is connected to the Internet via two links. One link is a $1000Mbps$ GigE fiber link provided by Verio, a commercial ISP, and the other connects to Internet 2. Our trace consists of the incoming and outgoing traffic collected on the first link using DAG cards made by Endace [4].

The traffic trace data set consists of seven hours' trace data collected from 9:30am to 10:30am in the morning for a week from July 16th to July 22nd, 2004. July 17th is a Saturday and July 18th is a Sunday. The hourly data traces taken on weekdays consist of approximately 3 to 4 Gigabytes data each, and the data traces taken on Saturdays and Sundays consists of approximately 1.5 to 2 Gigabytes data each.

For each packet, the first 52 bytes are captured and dumped to the hard disk. This includes 16 bytes in the ethernet header, 20 bytes IP header, and the next 16 bytes following the IP header. For TCP packets, this is the first 16 bytes in the TCP header, and for UDP packets, this includes the 4 bytes UDP header and another 12 bytes packet content.

All of these data are human labeled. We examine the traffic by collecting the flow and packet information. Anomalies are picked from flows with high volume of traffic, hosts with high numbers of incoming or outgoing flows, or port numbers that has high volume of traffic. It is not possible, however, to declare that all the anomalies in the data are correctly labeled. We only have access to the header of the packets, thus it is impossible to label some types of anomalies that can only be recognized from the content of the packets. Also the huge amount of data makes it impossible to examine every packet in the collected trace. Therefore, there may be anomalies missed in the human labeling process and anomalies missed by both human labeling and the anomaly detection approach.

B. Model training

We use the data taken on July $20th$ from 9:30am to 10:30am as the training data set. By human labeling, this set of data contains port scans or DOS attack at ports 1433, 135, 139, 445, 57, 5554, 9898, 5000, 81 and 32773 in different time periods. When training the model, these anomaly packets are not used.

The Maximum Entropy estimation algorithm described in Section IV-C is used to generate the baseline distribution of the packet classes from the training data. We set the stopping criterion for the induction algorithm to be whether the Kullback-Leibler difference of P with respect to $\tilde{P}$ is less than 0.01. By this criterion, the model induction algorithm ended with a set of 362 feature functions.

The first five features selected are $f_{TCP}(\omega)$, $f_{[80]}(\omega)$, $f_{[>49151]}(\omega)$, $f_{TCP\times[6824,6924]}(\omega)$, and $f_{[6324,6424]}(\omega)$. The first feature function selected, $f_{TCP}(\omega)$, shows that the dominant packets in the current traffic are TCP packets. The second feature function selected, $f_{[80]}(\omega)$, shows that the web traffic consists a majority part of the traffic. The third feature function selected, $f_{[>49151]}(\omega)$ reveals the fact that most applications, when connecting to remote hosts, choose a local port larger than 49151. The forth and the fifth feature functions selected show the popularity of peer to peer applications. The feature function $f_{TCP\times[6824,6924]}(\omega)$ shows the traffic of *Bit Torrent*, whose default ports range from 6881 to 6999, and the feature function $f_{[6324,6424]}(\omega)$ shows the traffic of *Gnutella*, whose default port is 6346.

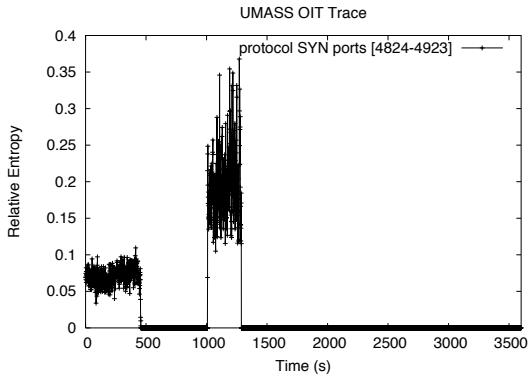

Fig. 2. Relative entropy for packets of type SYN and destination port number from 4824 to 4923

The training process is very effective. We use a 2.2GHz Dell workstation. It took less than two hours to finish the whole training process, including computing the empirical distribution from the data trace.

C. Detecting network anomalies

In this subsection, we describe some of the network anomalies that manifest themselves by increasing the $D_{\tilde{P}\|P}(\omega)$ value defined in (16), which is related to the relative entropy of the network traffic.

The parameters used in the experiments in this subsection and the following subsections are set as $\delta = 1$ second, $d = 0.01$, $w = 60$ and $h = 30$ unless otherwise specified. This is to say, if in a minute's time window, the entropy divergence of a class ω exceeds the threshold 0.01 for 30 times, an alarm is generated.

For the first example, we look at two cases of port scans. On July 19th, 2004, from 9:30am, when we began our data collection, to 9:37am, a host outside of the UMASS campus network performed a port scan at port 4899 by sending many SYN packets to different hosts in the UMASS campus network. The average scan rate was 107 packets/second and it lasted for at least 460 seconds. Then from 9:46am to 9:51am, another host outside of the UMASS campus network performed another port scan at the same port at an average rate of 232 packets/second. During these two time periods, the relative entropy of the packet class that represents SYN packets targeting at ports from 4824 to 4923 increased considerably, as shown in figure 2. From these figures, we can see that the more serious the port scan, the higher the relative entropy for the corresponding packet class. These two port scans were successfully detected by our relative entropy detection algorithm.

For the second example, we look at a SYN flooding attack. On July 21st, 2004, from 9:42am to 10:30am, when we stopped trace collection, a host outside of the UMASS campus performed an attack on a single host inside the UMASS campus by flooding SYN packets at the host's port 5100. At its peak of the attack, the sending rate of the SYN packets was 210 packets/second. Figure 3 shows the relative entropy of the packet class that corresponding to SYN packets targeted at port 5100. We see that there were two time periods in which the attack was most serious. This SYN flooding attack was also successfully detected by our relative entropy detection algorithm.

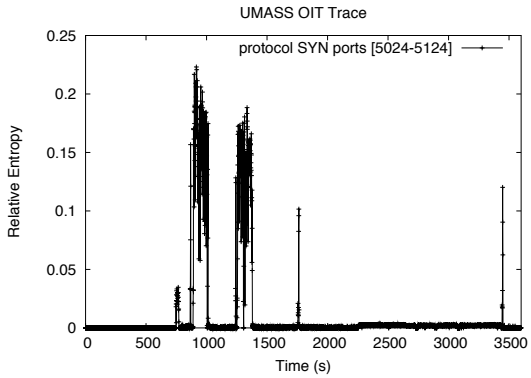

Fig. 3. Relative entropy for packets of type SYN and destination port numbers from 5024 to 5124

D. Algorithm performance

We test the performance of the algorithm by running it over the remaining six human labeled data sets. Again, we set $\delta = 1$ second, $d = 0.01$, $w = 60$ and $h = 30$. The detection algorithm will provide results at every time interval δ. If an anomaly is detected by the algorithm and there is a corresponding anomaly detected by human labeling, we call it a *positive*. All anomalies detected by the algorithm corresponding to the same anomaly labeled by human are treated as a single positive. If there is no human labeled anomaly correspond to the anomaly reported by the algorithm, the reported anomaly is called a *false positive*. Consecutive false positives are treated as a single false positive. Anomalies labeled by human but missed by the algorithm are called *false negatives*. The performance of the algorithm is evaluated based on the number of positives, false positives and false negatives. For the false positives, we revisit the traffic to see whether the anomaly detected by the anomaly detection approach is an anomaly missed by human labeling or not, and why the false positives occurred. If the reported anomaly is an anomaly missed by human, the human labeling data is updated accordingly and the corresponding 'false positives' are changed to positives.

The algorithm described in the previous section is executed on the remaining six traffic traces. In each case, the algorithm detects most of the anomalies located by human labeling. However, the algorithm also reports many anomalies that are not located by human labeling. We looked at these 'false positives' by tracing them back in the traffic trace. These 'false positives' mainly fall into three categories. The first category consists of the 'flash crowds' phenomenon. For example, we detect high volume 'eDonkey' traffic which uses TCP ports from 4661 to 4663. The second category consists of high rate traffic that communicates with port numbers rarely seen in the training data. In this category, we detect 'Quake II' users playing online games with UDP port 27910, and we detect a host inside the UMASS campus network that serves as an 'I*Net' data server, which uses UDP port 2441. For the third category, these are traffic that we cannot tell what they are, given the limited packet header information. Though these 'false positives' may still be interesting to the network administrators, we didn't treat them as anomalies.

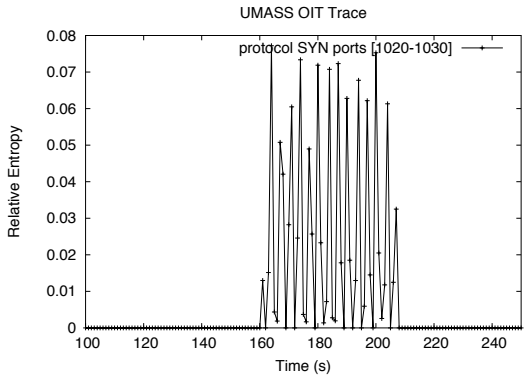

Fig. 4. Relative entropy for packets of type SYN and destination port number from 1020 to 1030

In spite of the ambiguous situation concerning all the anomalies generated by the algorithm, we found that the experimental results regarding SYN packets give good results. In the following, we look at experimental results regarding to SYN packets in each of the traffic traces.

In the data trace of July 16th, there are 10 anomalies detected by human labeling. The algorithm detected all of them, and in addition, the algorithm reported a false positive at the packet class corresponding to SYN packets with port numbers between 9424 to 9524. Upon reexamining the trace data, we see that this was caused by many hosts from outside the campus connecting to a host in the campus with the port 9493 which had a peak rate of 27 SYN packets per second. It is very likely that this is a flash crowd phenomenon caused by some P2P software such as bit torrent.

In the data trace of July 17th, there are 11 anomalies detected by human labeling. The algorithm detected 10 of them with no false positives. The one missed by the algorithm is a repeated connection attempt carried out by a single host outside the campus to a host inside the campus on port 3531. The algorithm missed this anomaly mostly because of the low rate of the anomaly, which is about 6 SYN packets per second.

In the data trace of July 18th, there are 14 anomalies detected by human labeling. The algorithm detected all of them with no false positives.

In the data trace of July 19th, there are 16 anomalies detected by human labeling. The algorithm detected 14 of them with no false positives. One of the anomalies missed by the algorithm is a port scan for port 1023. Actually this anomaly generated large $D_{\tilde{P}\|P}(\omega)$ values in its corresponding packet class, but since the attack was sometimes quite weak and lasted for a short period of time, the anomaly was ignored by the sliding window algorithm, as shown in Figure 4. Another anomaly missed by the algorithm is a port scan at port 21. Again, this anomaly generated large $D_{\tilde{P}\|P}(\omega)$ values, but was ignored by the sliding window algorithm as it was weak and lasted a very short time. We can see later in subsection VII-A that these two anomalies can both be detected by either decreasing the divergence threshold d, or decreasing h, the parameter used in the sliding window approach.

Date	Humanly labeled	Positive	False negative	False positive	Precision	Recall	F1
July 16	10	10	0	1	0.91	1	0.95
July 17	11	10	1	0	1	0.91	0.95
July 18	14	14	0	0	1	1	1
July 19	16	14	2	0	1	0.88	0.93
July 21	15	15	0	0	1	1	1
July 22	9	8	1	0	1	0.89	0.94

TABLE I
ALGORITHM PERFORMANCE

In the data trace of July 21st, there are 15 anomalies detected by human labeling. The algorithm detected all of them with no false positives.

In the data trace of July 22nd, there are 9 anomalies detected by human labeling. The algorithm detected 8 of them with no false positives. The anomaly missed is a 'vertical' port scan in which a host from outside the campus scanned all the ports of a single host inside the campus. Since this hardly caused any change in the packet distribution, it was missed by the relative entropy method.

Table I summarizes the algorithm performance of the algorithm in the experiments described above. The table also summarizes the performance of the algorithm in terms of precision, recall and F1. Let a be the number of positives, b be the number of false positives, c be the number of false negatives, precision is then defined as $a/(a+b)$, recall is defined as $a/(a+c)$ and F1 is defined as $2a/(2a+b+c)$. F1 measures a combination of precision and recall. From this table, we see that the relative entropy approach detects most of the anomalies located by human with rare false positives.

E. Comparison with Holt-Winter algorithm

In order to have some idea of what the data would be like under a change detection approach, we run the Holt-Winter algorithm[2] on the SYN traffic in each of the traffic collection above. We use the number of SYN packets observed in i seconds as the random variable that the Holt-Winter algorithm estimates and predicts, and we set $i = 1, 2, 3$. We observe that under the time scale of seconds, the Holt-Winter algorithm doesn't work well since the variance of the observed value is too high. Besides, when the Holt-Winter generates an anomaly, it cannot differentiate different types of anomalies and cannot identify the anomalies in the traffic. For experiments at a larger time scale, as the Holt-Winter algorithm considers seasonal factors which may correspond to different times of the day, we don't have a proper data set. We leave it as one of our future work.

VII. DISCUSSION

In this section we discuss several aspects of the work such as the choice of parameters and the training method. And we propose several improvements that might be beneficial to the current work.

δ	0.1s	0.5s	1.0s	1.5s	2.0s	5.0s	10.0s
Positive	15	15	14	13	13	13	12
False negative	1	1	2	3	3	3	4
False positive	7	1	0	0	1	1	0

TABLE II

ALGORITHM PERFORMANCE UNDER DIFFERENT δ

d	0.001	0.002	0.005	0.01	0.02	0.05	0.1	0.2
Positive	16	16	14	14	14	12	8	1
False negative	0	0	2	2	2	4	8	15
False positive	12	8	2	0	0	0	0	0

TABLE III

ALGORITHM PERFORMANCE UNDER DIFFERENT d

A. Parameter sensitivity

In the previous sections, the parameters used are chosen in an ad hoc way. In this subsection, we discuss the sensitivity of the algorithm performance with regards to the parameters chosen. Three parameters are covered in three sets of experiments: the time interval δ, the divergence threshold d, and the parameter h used in the sliding window approach described in Section V, which controls how many times of exceeding the divergence threshold would cause an anomaly. In each set of experiments, we fix other parameters as used in the pervious experiments and change only the parameter under discussion. We show only the experimental results from the data set collected in July 19th. Experiments on other data sets show similar results.

Table II shows the sensitivity of the algorithm performance with regard to the time interval δ. With other parameters fixed as before, we changed δ from 0.1 second to 10.0 second. As δ increases, the number of packets involved in computing the divergence increases. In this case, the distribution of the traffic under observation becomes more similar to that of the baseline distribution and the anomalies are more likely to be smoothed out. Thus, as δ increases, fewer anomalies are detected, and, for the same reason, fewer false positives as well. We also observe that as δ increases, the number of positives decrease slowly and there are not many false positives when δ is small.

Table III shows the experimental results under different choices of the divergence threshold d. With other parameters fixed as before, we changed d from 0.001 to 0.2. As d increases, the algorithm becomes less and less sensitive to the difference between the distribution of the traffic under observation and the baseline distribution. This explains the decrease in the numbers of detected anomalies and false positives.

Table IV shows the experimental results under different choices of the parameter h. With other parameters fixed as before, we changed h from 15 to 45. Increasing h corresponds to being less sensitive to the divergence, which explains the decrease in the numbers of detected anomalies and false positives. Also we see that the decrease in

h	15	20	25	30	35	40	45
Positive	16	16	15	14	13	13	13
False negative	0	0	1	2	3	3	3
False positive	1	1	1	0	0	0	0

TABLE IV

ALGORITHM PERFORMANCE UNDER DIFFERENT h

	Training	Humanly labeled	Positive	False negative	False positive	Precision	Recall	F1
July 16	Supervised	10	10	0	1	0.91	1	0.95
	Unsupervised	10	6	4	1	0.86	0.60	0.71
July 17	Supervised	11	10	1	0	1	0.91	0.95
	Unsupervised	11	9	2	0	1	0.82	0.90
July 18	Supervised	14	14	0	0	1	1	1
	Unsupervised	14	12	2	0	1	0.86	0.92

TABLE V

ALGORITHM PERFORMANCE WITH AND WITHOUT HUMAN LABELED TRAINING DATA

positives is slow and there are not many false positives connected with small h values. Note also that the two anomalies previously missed by the algorithm are both detected in this case and the previous case.

In summary, the above experimental results show that the relative entropy detection approach is pretty robust to the choice of parameters. The choice of d seems to be the most critical to the performance of the algorithm. Nevertheless, the algorithm still gives high positives and low false positives in a large range of parameter settings in each set of the experiments.

B. Supervised vs. unsupervised training

The experimental results shown in the previous section are based on a baseline distribution obtained from a set of human labeled training data. Removing the anomaly data from the training set helps to build a 'cleaner' model that better identifies anomalies in the observed/testing data. However, completely labeling the data is sometimes impossible under such constraints as lack of human power or privacy concerns. In these situations, a more or completely automated process, i.e. with less or no human interference like data labeling, is more preferable. Towards that end, we redo all the experiments with a baseline distribution obtained from the same training data but without using any labeling information. We then compare the performance of the experiments using labeled data with that using unlabeled data. It is expected that as anomaly data is included into the training data, the baseline model will be more tolerant to the existence of anomalies in the traffic. Therefore, the number of positives is expected to decrease, and the number of false negative increase.

Table V lists the experimental results of 3 days' data based on baseline model trained from data with human label and without human label. Other experiments show similar results. These results show that, with unlabeled

training data, the detection algorithm is able to report most of the anomalies detected by human labeling without significant increase in the number of false positives. It implies that even with no human labeling at the training process, we can still obtain a fairly good anomaly detection baseline distribution. An interesting application of this is to build a 'cleaner' baseline distribution without human interference, which we will cover in the next subsection.

C. Extensions and future work

Several possible extensions can potentially enhance our current approach. The experiments shown in this work are based on the two-dimensional classification of IP packets described in Section III. A more general higher-dimensional classification of the packets would provide a more thorough view of the packets in the traffic, and more interesting anomalies in the network can then be revealed. ICMP packets, for example, can be added to the packet classes. This would reveal anomalies such as ping attacks which the current approach cannot detect. The IP addresses can also be added as a dimension of the packet class. This helps to detect anomalies such as 'vertical' port scans that scan a single host on multiple ports. The vertical port scans change the number of packets sent to the destination host. With the current approach, it is difficult to detect such anomalies as they usually do not change the portions of packets with particular protocols or port numbers.

The time information can also be added as one of the dimensions when creating the packet classes. In different times of the day, the network traffic have different packet distributions. By adding the time as one of the dimensions in the packet class and using a set of traffic traces that are collected during different times of the day, it is possible to represent the time variant packet distribution with a few parameters using the Maximum Entropy framework. Thus a single model can be used as the baseline distribution at different times of the day. It would also be interested to compare the results with other change detection algorithms such as Holt-Winter.

Another interesting future work is to automate the training process. The previous subsection shows that with a baseline distribution trained from unlabeled data we can still detect most of the anomalies in the traffic. Therefore, with the baseline distribution we can detect anomalies existing in the training data. By removing these anomalies, we can get a 'cleaner' baseline distribution. This process might be iterated until no anomalies is found in the training data using the obtained baseline distribution. It would be interesting to know the performance reached using this approach versus that using human labeling.

VIII. Conclusion

In this paper, we introduce our approach to detect anomalies in the network traffic using the relative entropy information. The packet distribution in the network traffic is estimated using the Maximum Entropy framework and used as a baseline to detect the anomalies. The method is able to detect anomalies by inspecting only the current traffic instead of a change point detection approach. The experimental results show that it effectively

detects anomalies in the network traffic including different kinds of SYN attacks and port scans. This anomaly detection method identifies the types of the anomalies detected and comes with low false positives. It is easy to deploy and very scalable. Many interesting aspects of this approach still remain to be explored.

REFERENCES

[1] BARFORD, P., KLINE, J., PLONKA, D., AND RON, A. A signal analysis of network traffic anomalies. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop* (2002).

[2] BRUTLAG, J. D. Aberrant behavior detection in time series for network service monitoring. In *Proceeding of the 14th Systems Administration Conference* (2000), pp. 139–146.

[3] DARROCH, J. N., AND RATCLIFF, D. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics 43*, 5 (1972), 1470–1480.

[4] ENDACE. http://www.endace.com.

[5] INTERNET ASSIGNED NUMBERS AUTHORITY. http://www.iana.org/assignments/port-numbers.

[6] LEE, W., AND XIANG, D. Information-theoretic measures for anomaly detection. In *Proceedings of the IEEE Symposium on Security and Privacy* (2001), IEEE Computer Society, p. 130.

[7] MALOUF, R. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conference on Natural Language Learning* (2002).

[8] MCCALLUM, A. Efficiently inducing features of conditional random fields. In *Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI03)* (2003).

[9] PAXSON, V. Bro: A system for detecting network intruders in real-time.

[10] PIETRA, S. D., PIETRA, V. D., AND LAFFERTY, J. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 4 (1997), 380–393.

[11] SNORT: THE OPEN SOURCE NETWORK INTRUSION DETECTION SYSTEM". http://www.snort.org/.

[12] STANIFORD, S., HOAGLAND, J., AND MCALERNEY, J. M. Practical automated detection of stealthy portscans. In *Proceedings of the IDS Workshop of the 7th Computer and Communications Security Conference* (2000).

[13] THOTTAN, M., AND JI, C. Anomaly detection in ip networks. *IEEE Trans. Signal Processing 51* (2003).

[14] WANG, H., ZHANG, D., AND SHIN, K. G. Detecting syn flooding attacks. In *Proceedings of IEEE INFOCOM* (2002).