

# Detecting Denial-of-Service Attacks with Incomplete Audit Data

Animesh Patcha and Jung-Min Park  
Bradley Department of Electrical and Computer Engineering  
Virginia Polytechnic Institute and State University  
Blacksburg, Virginia 24061  
Email: {apatcha, jungmin}@vt.edu

**Abstract**—With the ever increasing deployment and usage of gigabit networks, traditional network anomaly detection based Intrusion Detection Systems have not scaled accordingly. Most, if not all, systems deployed assume the availability of complete and clean data for the purpose of intrusion detection. We contend that this assumption is not valid. Factors like noise in the audit data, mobility of the nodes and the large amount of network data generated by the network make it difficult to build a normal traffic profile of the network for the purpose of anomaly detection. From this perspective, we present an anomaly detection scheme, called SCAN (Stochastic Clustering Algorithm for Network anomaly detection), that has the capability to detect intrusions with high accuracy even when audit data is not complete. We use the Expectation–Maximization algorithm to cluster the incoming audit data and compute the missing values in the audit data. We improve the speed of convergence of the clustering process by using Bloom filters and data summaries. We evaluate SCAN using the 1999 DARPA/Lincoln Laboratory intrusion detection evaluation dataset.

## 1. Introduction

With the Internet having evolved in leaps and bounds over the past decade, most Intrusion Detection Systems (IDS) have not been able to keep up with the advances in high speed networking. IDS products, currently deployed in gigabit networks, need significant improvements before they can offer adequate protection against attacks. A majority of the products in the market today can detect less than half of the attacks directed at them, even though many of those attacks are well documented [18].

In this paper, our focus is on network anomaly detection in a large, high volume and high speed enterprise network. Our particular interest is on the detection of network anomalies when complete network audit data is not available. An anomaly detection system builds a normal profile of the network and then detects deviations from normal usage patterns. Traditionally anomaly detection systems have relied on the availability of *clean* and *complete* data for analysis. This, however, is not a valid assumption. For example, in high speed gigabit networks it is impractical to store, classify and label network traffic to create a normal profile because of the large volume of network traffic. Another example can be seen in Mobile and Ad hoc Networks (MANETs). Mobility, sleep patterns of mobile devices and other characteristics unique to MANETs, prevent the collection of complete network data for analysis.

In this paper, we describe and evaluate SCAN (Stochastic Clustering Algorithm for Network anomaly detection), an unsupervised network anomaly detection scheme. At the core of SCAN is a stochastic clustering algorithm which is based on an improved version of the Expectation–Maximization (EM) algorithm [5]. The EM algorithm’s

speed of convergence was accelerated by using a combination of Bloom filters, data summaries and associated arrays. The clustering approach that we propose can be used to process incomplete and unlabelled data. To the best of our knowledge, there have been no prior attempts to investigate the effects of incomplete datasets in anomaly detection. The salient features of SCAN are:

- *Improvements in speed of convergence*: We use a combination of data summaries, Bloom filters and arrays to accelerate the rate of convergence of the EM algorithm.
- *Ability to detect anomalies in the absence of complete audit data*: We use SCAN to detect anomalies in network traffic in the absence of complete audit data. The improved EM clustering algorithm—which forms the core of SCAN—is used to compute the missing values in the audit dataset. The advantage of using the EM algorithm is that unlike other imputation techniques, EM computes the maximum likelihood estimates in parametric models based on prior information.

The remaining parts of the paper are organized as follows. In Section 2, we review related work. Section 3 describes the system model and the clustering algorithm used by SCAN. In Section 4, we evaluate the performance of SCAN. Section 5 concludes the paper.

## 2. Related Work

Network anomaly detection systems, such as NIDES [1], learn a statistical model of normal network traffic and flag deviations from this model. NIDES uses a frequency-based model in which the probability of an event is estimated by its average frequency during training. The model is based on the distribution of source and destination IP addresses and ports per transaction. NIDES models ports and addresses, flagging differences between short and long term behavior.

PHAD (Packet Header Anomaly Detector) [12], LERAD (LEarning Rules for Anomaly Detection) [13] and ALAD (Application Layer Anomaly Detector) [14] use time-based models in which the probability of an event depends on the time since it last occurred. For each attribute, the aforementioned schemes collect a set of allowed values and flag novel values as anomalous. PHAD, ALAD, and LERAD differ in the attributes that they monitor. PHAD monitors 33 attributes from the Ethernet, IP and transport-layer packet headers. ALAD models incoming server TCP requests: source and destination IP addresses and ports, opening and closing TCP flags, and the list of commands (the first word on each line) in the application payload. Depending on the attribute, it builds separate models for each target host, port number (service), or host/port combination. LERAD also models TCP connections.

In network intrusion detection, the network traffic that is available is primarily categorical<sup>1</sup>. Many of the clustering approaches that process categorical data, such as ROCK [8], are often resource intensive. For example, ROCK has a high computational cost and requires sampling in order to scale to large datasets. Unsupervised approaches<sup>2</sup> for detecting outliers in large dataset have been proposed [9], [17], but these approaches are primarily based on ordered data.

The idea to improve EM to work with large datasets is not new. The most important previous work comes from the machine learning community. Neal *et al.* [15] introduce the idea of using sufficient statistics<sup>3</sup>. In this work, the authors analyze several variants of EM, conceptualizing the logarithmic-likelihood optimization as an optimization of an energy function. The key idea behind one of the variants presented in their work is to use sufficient statistics. They describe a simple mixture problem with a dimensionality of one and a cluster size of two. They also show experimental evidence of the superiority of their approach. However, their study is incomplete as it does not address the problem of clustering large datasets with high dimensionality. Problems related to numerical stability, proper initialization, and outlier handling are not addressed in their work either. In [3], Bradley *et al.* present a scalable EM (SEM) algorithm that iterates in memory and also summarizes points through their sufficient statistics. To avoid locally optimal solutions, they re-seed empty clusters and estimate several models concurrently, sharing information across models. The scheme presented by the authors, requires the user to specify what fraction of the working buffer should be from the entire database. The value for this parameter is typically 1%. The authors, however, do not explain why this value gives the best results. BIRCH [19] and newer extensions of the classical EM algorithm [3], [16] use data summarization to accelerate convergence.

### 3. System Model and Design

In this section, we describe the system model (see Fig. 1) and outline the design philosophy behind SCAN. SCAN has three major components:

- 1) Online Sampling and Flow Aggregation
- 2) Clustering and Data Reduction
- 3) Anomaly Detection

We describe each unit in detail below.

#### 3.1 Online Sampling and Flow Aggregation

In this module, network traffic is sampled and classified into flows. In the context of SCAN, a *flow* is all the connections with a particular destination IP address and a port number combination. The measurement and characterization of the sampled data proceeds in time slices. We process the sampled packets into *connection records*. Because of its compact size as compared to other types of records (e.g., packet logs), connection records are appropriate for data analysis. Connection records provide the following fields:

(*SrcIP, SrcPort, DstIP, DstPort, ConnStatus, Proto, Duration*)

Each field can be used to correlate network traffic data and extract intrinsic features of each connection.

<sup>1</sup>Categorical data is data that fits into a small number of discrete categories.

<sup>2</sup>A learning approach in which the system parameters are adapted using only the information of the input and are constrained by pre-specified internal rules. It is distinguished from a supervised approach by the fact that there is no *a priori* output.

<sup>3</sup>A quantity  $T(X)$  that depends on the (observable) random variable  $X$  but not on the (unobservable) parameter  $\theta$  is called a statistic. A sufficient statistic captures all of the information in  $X$  that is relevant to the estimation of  $\theta$ .

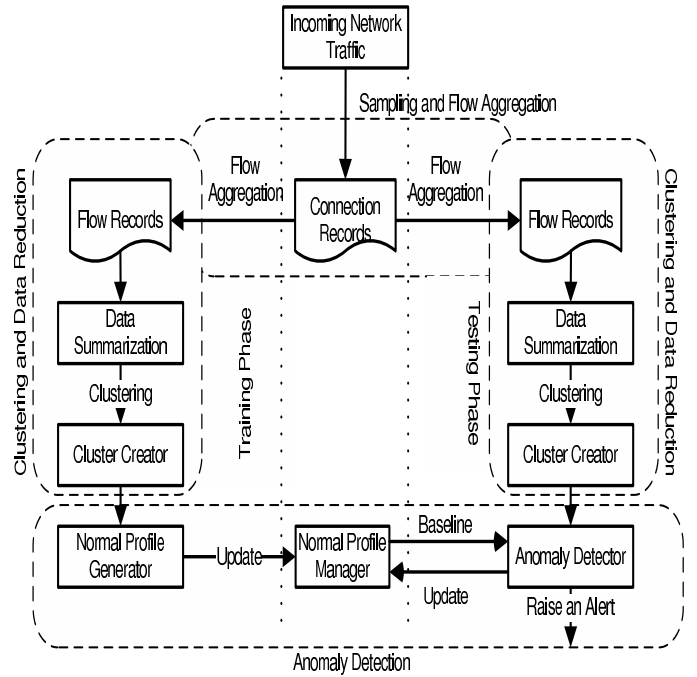


Fig. 1: System model.

We employ a very simple flow aggregation algorithm (see Fig. 2). Bloom filters [2] and associated arrays are used to store, retrieve and run membership queries on incoming flow data in a given time slice. When a packet is sampled, we first identify the flow the packet belongs to by executing a membership query in the Bloom filter based on the *flow ID*<sup>4</sup>. If an existing *flow ID* cannot be found, a new *flow ID* is generated based on the information from the connection record. This is called the *insertion phase*. If the *flow ID* exists, the corresponding flow array is updated. If the *flow ID* does not exist, we run the *flow ID* through each of the  $k$  hash functions (that form a part of the Bloom filter), and use the result as an offset into the bit vector, turning on the bit we find at that position. If the bit is already set, we leave it on.

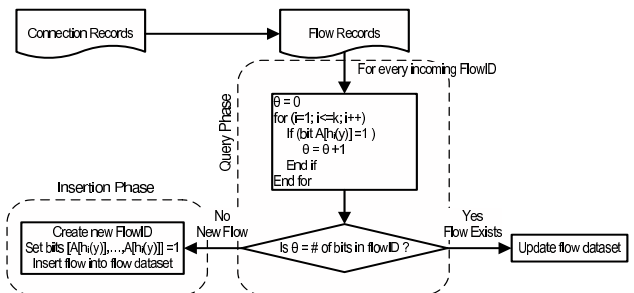


Fig. 2: Online sampling and flow aggregation algorithm.

Bloom filters were first proposed in [2] as a space efficient data structure for answering approximate membership queries over a given set. One drawback of Bloom filters is the risk of false positives (i.e., returning a “yes” indicating

<sup>4</sup>The *flow ID* is a unique ID that is generated from a combination of the destination IP address and the destination port number.

that a given element is in the set when it is not). The false positive probability is a function of the length of the filter and the number of items stored in it.

To succinctly represent a set  $S = \{x_1, x_2, \dots, x_n\}$  of  $n$  elements and support membership queries, we first select an array  $A$  consisting of  $m$  1-bit elements and initialize it to zero. In addition, we select  $k$  distinct hash functions  $h_1, \dots, h_k$  each with a range of  $[1, m]$ . During the *insertion phase*, for each element  $x \in S$ , the bits at the position  $h_1(x), \dots, h_k(x)$  in  $A$  are set to 1. In the *query phase*, to check if an element  $y$  is in  $S$ , we check the value of the bits at positions  $h_1(y), \dots, h_k(y)$  in  $A$ . The answer to the query is a *yes* if all of these bits are one and *no* otherwise.

### 3.2 Clustering and Data Reduction

The goal of clustering is to divide flows into natural groups. The instances contained in a cluster are considered to be similar to one another according to some metric based on the underlying domain from which the instances are drawn. A stochastic clustering method (such as the EM algorithm) assigns a data point to each group with a certain probability. Such statistical approaches make sense in practical situations where no amount of training data is sufficient to make a completely firm decision about cluster memberships.

Stochastic clustering algorithms aim to find the most likely set of clusters given the training data and prior expectations. The underlying model, used in the stochastic clustering algorithm, is called a *finite mixture model*. A mixture is a set of probability distributions—one for each cluster—that models the attribute values of the members of that cluster.

In the finite mixture model under consideration, we consider a scenario where we assume that the distribution  $\mathbf{T}$  that describes incoming network traffic<sup>5</sup> to a server  $s$  is distributed as a mixture of two populations [6]: a *normal* ( $\mathbf{N}$ ) distribution and an *anomalous* ( $\mathbf{A}$ ) distribution such that the  $i^{\text{th}}$  traffic element<sup>6</sup>,  $x_i$ , in a given time slice  $t$  is generated from the *normal* distribution  $\mathbf{N}$  with a probability of  $p$  or from an *anomalous* distribution  $\mathbf{A}$  with a probability of  $1 - p$ . In other words,

$$\mathbf{T} = p\mathbf{N} + (1 - p)\mathbf{A}. \quad (1)$$

During the training phase, we set the baseline activity profile assuming that all incoming traffic elements are generated from the normal distribution  $\mathbf{N}$ .

#### 3.2.1 EM Algorithm-Based Clustering Algorithm

We use the EM algorithm (see Fig. 3) to cluster data. The EM algorithm is a general method of finding the maximum-likelihood estimate of the parameters of a distribution from a given data set when the dataset has missing values. For this discussion, let us assume that we have a density function  $P(Z|\Theta)$  that is governed by the set of parameters  $\Theta$ . In addition, we have a dataset of size  $n$  drawn from this distribution,  $Z = \{z_1, \dots, z_n\}$ . The function  $L(\Theta|Z)$  is called the likelihood function and is defined as the likelihood of the parameters given the dataset, i.e.,

$$L(\Theta|Z) = P(Z|\Theta) = \prod_{i=1}^n P(z_i|\Theta) \quad (2)$$

<sup>5</sup>It should be pointed out here that the incoming traffic could be composed of either packets, flows or connections.

<sup>6</sup>The term *traffic element* is used to denote any of the elements in one connection record.

In a maximum likelihood problem, our goal is to find  $\Theta$  that maximizes  $L$ .

The EM algorithm starts with an initial guess for the parameters of the mixture model for each cluster and then iteratively applies a process, viz., the *Expectation Step* ( $E$ ) and the *Maximization Step* ( $M$ ), in order to converge to the maximum likelihood fit. In the  $E$ -step, the EM algorithm first finds the expected value of the complete-data log-likelihood, given the observed data and the parameter estimates. In the  $M$ -step, the EM algorithm maximizes the expectation computed in the  $E$ -step. The two steps of the EM algorithm

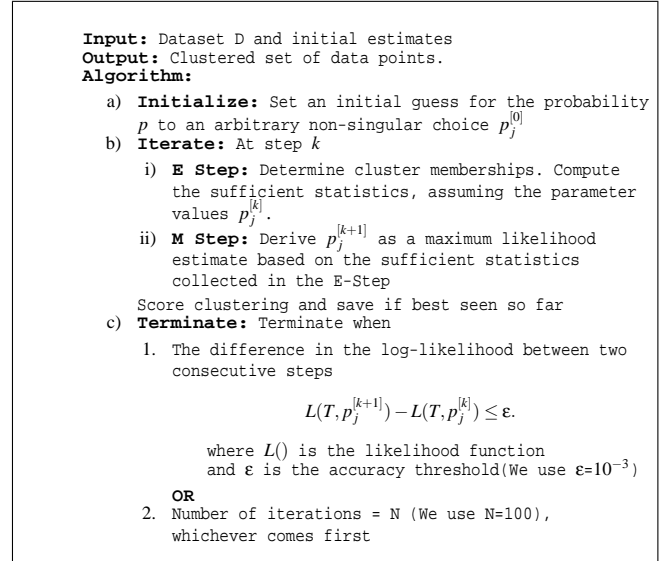


Fig. 3: EM algorithm for clustering.

are repeated until an increase in the log-likelihood of the data, given the current model, is less than the accuracy threshold  $\epsilon$ .

The EM algorithm is *guaranteed* to converge to a local maximum which may or may not be the same as the global maximum. Typically, the EM algorithm is executed multiple times with different initial settings for the parameter values. Then a given data point is assigned to the cluster with the largest log-likelihood score.

The classical EM algorithm does have a few disadvantages. In general, the algorithm is hard to initialize and the quality of the final solution depends on the quality of the initial solution. It may also converge to a poor locally optimal solution. This means that a solution may be acceptable, but is far from the optimal one. The EM algorithm needs an unknown number of iterations to converge to a good solution. That is, it is hard to set a threshold on the maximum number of iterations that works. The EM algorithm, therefore, usually requires several passes over a dataset.

#### 3.2.2 Data Summaries for Data Reduction

Our aim was to design a clustering algorithm which is fast and requires only a few passes over the data. The data space is assumed to contain data points with both numerical and categorical attributes. The attributes of each data point are the *words* from the corresponding audit file line. Note that we are using the term *word* to loosely describe a monitored attribute (e.g., IP addresses, port numbers, etc.). From the flow records that have been hashed and stored in the Bloom filters and the associated arrays, we calculate *data*

summaries (see Fig. 4) for every time slice. Data reduction techniques like sufficient statistics and data summaries are often used to enhance the speed of convergence of the EM algorithm [16].

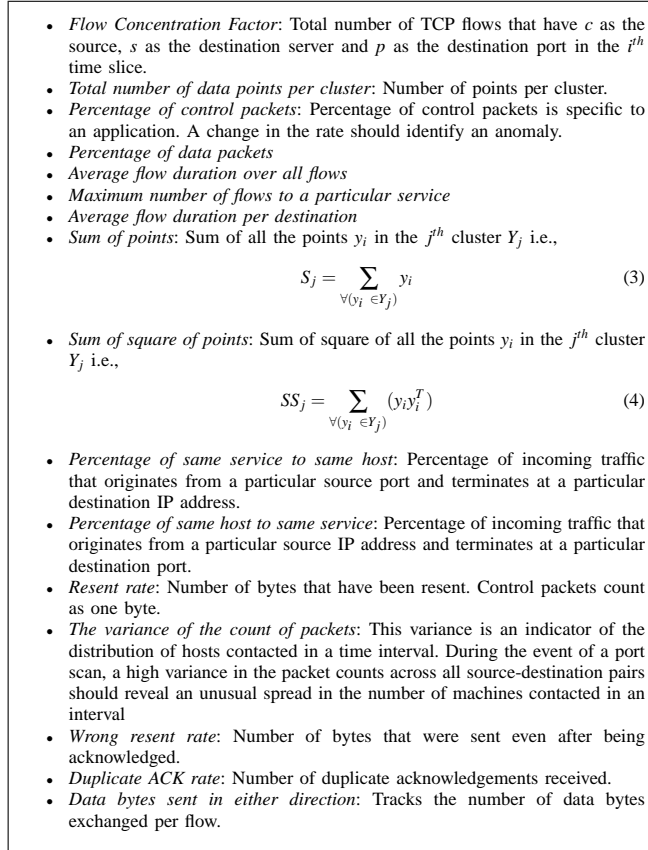


Fig. 4: Data summaries.

Our enhancements to the EM algorithm consist of three stages:

- **Stage I**: At the end of each time slice, we make a pass over the connection record dataset and build data summaries. Data summaries are particularly important in the EM algorithm framework as they reduce the I/O time by avoiding repeated scans over the data. In addition, data summaries allow periodic parameter estimation as the records are being read. This enhances the speed of convergence of the EM algorithm.
- **Stage II**: We make another pass over the dataset to build cluster candidates using the data summaries collected in the first stage. After the frequently appearing words have been identified, the algorithm builds all cluster candidates during this pass. The cluster candidates are hashed and stored in Bloom filters and associated arrays. The dataset is processed line by line. When a line is found to belong to one or more dense regions (i.e., one or more frequently appearing words have been discovered in the line), a cluster candidate is formed. This is the E-step of the EM algorithm, where the cluster membership is determined. If the cluster candidate is not present in the Bloom filter, it is inserted into the filter with the value one. If the cluster candidate is present in the Bloom filter, the corresponding value in the array is incremented
- **Stage III**: Clusters are selected from the set of candidates.

Learning steps (periodic M-steps) are used to accelerate convergence. By using data summaries, the clustering algorithm can run the M-step at different frequencies while scanning the connection records. We can minimize the convergence time by running the M-step after every point. Unfortunately, the EM algorithm would not produce the globally optimal solution in such a scenario. On the other hand, we can produce a solution that is closer to the globally optimum solution by executing the M-step after all the  $n$  points are read. This is, however, identical to the classical EM algorithm and the convergence time would not be reduced. Therefore, it is desirable to set the frequency of M-step execution in between the two aforementioned cases, but closer to the latter scenario. When a good approximation to the solution has been reached, we can execute normal EM iterations until the algorithm converges. During the final step of the clustering algorithm, the Bloom filter and the associated arrays are inspected, and the regions with values equal or greater than the threshold value are reported by the algorithm as clusters.

### 3.2.3 Handling Missing Data in a Dataset

The most appropriate way to handle incomplete datasets can be determined if one can determine how the missing data points got lost in the first place. Little *et al.* [10] classify missing data into three categories: Missing Completely at Random (MCAR), Missing at Random (MAR) and non-ignorable.

The technique of using the EM algorithm to handle missing data is documented extensively in [10]. In the E-step, SCAN computes the expected value of the complete data's log-likelihood value based on the complete data cases and the clustering algorithm's "best guess". The actual imputed values for the missing data points are not generated. In the M-step, the clustering algorithm inserts the expected values obtained from the E-step in place of the missing data and then maximizes the likelihood function to obtain new parameter estimates. The new parameter estimates are substituted back into the E-step and a new M-step is performed. The procedure iterates through these two steps until convergence is obtained. Convergence occurs when the difference in the parameter estimates from one iteration to another becomes less than the accuracy threshold  $\epsilon$ . The EM algorithm-based technique is purportedly the best maximum likelihood estimation technique for dealing with missing data [11].

### 3.3 Anomaly Detection

The first task in an anomaly detection process is *network baselining*. Network baselining can be defined as the act of measuring and rating the performance of a network. Providing a network baseline requires evaluating the normal network utilization, protocol usage, peak network utilization, and average throughput of the network usage over a period of time. Our approach to network baselining recognizes the fact that any parametric model of network traffic is an approximation to reality. Since our ultimate goal is anomaly detection, which we formulate as detecting a marked change in the baseline model parameters, we will not be solving the more general problem of parameter estimation as an intermediate step to anomaly detection.

In our model, elements of the network flow that are anomalous are generated from the anomalous distribution (A), and normal traffic elements are generated from the normal distribution (N). Therefore, the process of detecting

anomalies involves determining whether the incoming flow belongs to distribution **A** or distribution **N**. We use an approach similar to [6] and calculate the likelihood of the two cases to determine the distribution to which the incoming flow belongs to. We assume that for normal traffic, there exists a function  $F_N$  which takes as parameters the set of normal elements  $N_t$ , at time  $t$ , and outputs a probability distribution  $P_{N_t}$  over the data  $T$  generated by distribution **T**. Similarly, we have a function  $F_A$  for anomalous elements which takes as parameters the set of anomalous elements  $A_t$  and outputs a probability distribution  $P_{A_t}$ . Assuming that  $p$  is the probability with which an element  $x_t$  belongs to **N**, the likelihood  $L_t$  of the distribution **T** at time  $t$  is defined as

$$\begin{aligned} L_t(\mathbf{T}) &= \prod_{i=1}^n P_T(x_i) \\ &= \left[ (p)^{|N_t|} \prod_{x_i \in N_t} P_{N_t}(x_i) \right] \left[ (1-p)^{|A_t|} \prod_{x_j \in A_t} P_{A_t}(x_j) \right] \end{aligned} \quad (5)$$

The likelihood values are often calculated by taking the natural logarithm of the likelihood rather than the likelihood itself because likelihood values are often very small numbers that are close to zero. We calculate the log-likelihood as follows:

$$\begin{aligned} LL_t(\mathbf{T}) &= |N_t| \ln(p) + \sum_{x_i \in N_t} \ln P_{N_t}(x_i) \\ &\quad + |A_t| \ln(1-p) + \sum_{x_j \in A_t} \ln P_{A_t}(x_j) \end{aligned} \quad (6)$$

We measure the likelihood of each flow,  $x_t$ , belonging to a particular group by comparing the difference  $LL_t(\mathbf{T}) - LL_{t-1}(\mathbf{T})$ . If this difference is greater than some value  $\alpha$ , we declare the flow anomalous. We repeat this process for every flow in the time interval. Note that we have to recompute the probability distributions  $P_{N_t}(x_t)$  and  $P_{A_t}(x_t)$  at every step because of changes in their distributions.

## 4. Simulation Results

We evaluated SCAN using the 1999 DARPA intrusion detection evaluation data. The TCPDump files from the DARPA dataset consist of hundreds of thousands of lines of network data. Perl scripts were used to parse the TCPDump files into connection records. The connection records were then used to build the data summaries at the end of every time interval. We also aggregated the incoming network traffic into flows based on the flowID, and calculated various network parameters (see Fig. 4) for every time interval (duration of a time interval is 60 seconds). The resulting flow records and the data summaries were stored in plain text files, which were then used as inputs to the clustering algorithm. SCAN was implemented in the *MATLAB* environment.

### 4.1 Simulation Experiment Setup

Before we present the details of the simulation results, we would like to set forth three important assumptions that we have made. The primary assumptions of intrusion detection are: user and/or network activities are observable and more importantly, normal and intrusive activities have distinct behavior. In addition, we also assume that anomalous activity forms a small percentage of total network activity.

We use the DARPA dataset to evaluate SCAN. The DARPA dataset, despite its shortcomings, has been used

extensively in IDS research and is one of the few openly available datasets. The dataset consists of five weeks of TCPDump data. Data from week 1 and 3 consist of normal attack-free network traffic. Week 2 data consists of network traffic with labelled attacks. The week 4 and 5 data are the ‘‘Test Data’’ and contain 201 instances of 58 different unlabelled attacks, 177 of which are visible in the inside TCPDump data<sup>7</sup>. We trained SCAN on the DARPA dataset using week 1 and 3 data, then evaluated the detector on weeks 4 and 5. Evaluation was done in an off-line manner. For our experiments, we only used the inside TCPDump data. The 1999 DARPA dataset has five classes of attacks, viz., probes, denial-of-service (DoS) attacks, user-to-root attacks, remote-to-local attacks, and data attacks. In this paper, we focus on DoS attacks.

For the purpose of anomaly detection, we first process the data as mentioned in Section 3.1. Thereafter, we apply the SCAN clustering technique to the DARPA dataset. During the *training phase* we set up a baseline of normal activity. This is achieved by feeding the flow records and data summaries, obtained after processing the *normal* TCPDump data, as inputs to the clustering algorithm. The cluster boundaries that are obtained at the conclusion of this phase gives us the ‘‘boundary’’ of a normal cluster.

The cluster centers, covariance matrices, and probabilities for each cluster that are obtained during the *training phase* are used as baseline inputs when SCAN is brought online. This is the *testing phase*. As incoming network data is processed and clustered by SCAN, any points that lie beyond a cluster boundary are termed as *outliers*. Based on the tolerance level of the IDS and the relative distance of the outliers from the cluster centers, the outliers will be labelled as normal points or anomalies. In our simulations, we assume that the tolerance level is zero, that is, during the *testing phase*, any point that lies outside a cluster boundary is considered an anomaly.

### 4.2 Simulation Results with Complete Audit Data

An IDS is evaluated on the basis of accuracy and efficiency. To judge the efficiency and accuracy of SCAN, we use Receiver-Operating Characteristic (ROC) curves. An ROC curve, which graphs the *rate of detection* versus the *false positive ratio*, is one of the most widely used indicators of performance in evaluating an IDS. The *false positive ratio* is defined as the fraction of normal traffic (in our case, the fraction of all network flows) that is wrongly declared anomalous.

The performance of SCAN at detecting a SSH Process Table attack is shown in Fig. 5. The attack is similar to the Process Table attack in that the goal of the attacker is to cause the *SSHD* daemon to spawn the maximum number of processes that the operating system will allow.

In Fig. 6 the performance of SCAN at detecting a SYN flood attack is evaluated. A SYN flood attack is a type of a DoS attack. It occurs when an attacker makes a large number of half-open connections to a given port of a server during a short period of time. This causes the data structure in the ‘tcpd’ daemon in the server to overflow. Due to the buffer overflow, the system will be unable to accept any new incoming connections till the queue empties.

We compared the performance of SCAN with that of the *K-Nearest Neighbors* (KNN) clustering algorithm proposed

<sup>7</sup>The inside TCPDump data is the data collected by a sniffer located inside a simulated military base.

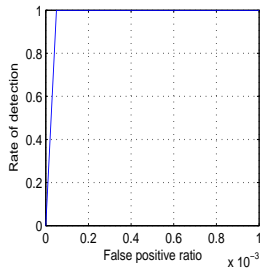


Fig. 5: ROC curve for SSH Process Table attack detection.

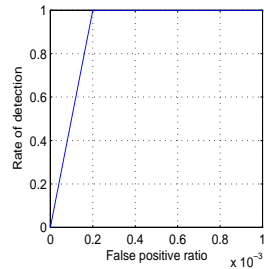


Fig. 6: ROC curve for SYN Flood (Neptune) attack detection.

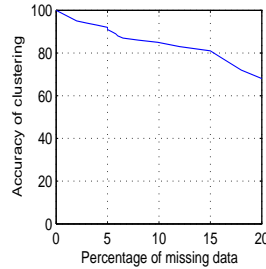


Fig. 7: Accuracy of clustering vs percentage of missing data.

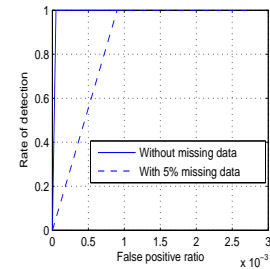


Fig. 8: ROC curve for SYN Flood (Neptune) attack detection with and without missing data.

in [7]. To classify a new datapoint, the KNN algorithm finds the  $K$ -nearest neighbors among the training data, and uses the categories of the  $K$ -nearest neighbors to weight the category candidates. The KNN method assumes all instances correspond to points in an  $n$ -dimensional space. The nearest neighbors of an instance are defined in terms of the standard Euclidean distance. Comparisons of the ROC curves seen in Fig. 5 and 6 with the results obtained in [7] suggests that SCAN outperforms the KNN algorithm. The two techniques are similar to each other in that both are unsupervised techniques that work with unlabelled data.

### 4.3 Effect of Missing Data

In the simulations, we assume that the missing data are missing at random (MAR). To introduce  $N\%$  missing features to a data set of  $D$  records, each of which has  $F$  features, we select randomly  $\lfloor (N \times D \times F) / 100 \rfloor$  records. For each selected record, we delete a randomly chosen feature to obtain a MAR dataset.

Fig. 7 shows the accuracy of clustering<sup>8</sup> versus percentage of missing data. It can be seen that even with 10% missing data, the accuracy of clustering is in the high eighties. As expected, we see that the accuracy of clustering as well as the performance of the anomaly detection scheme degrades with increasing percentage of missing data. In Fig. 8, we plot SCAN's ROC curves for when complete audit data is available and for when 5% of the audit data is missing.

## 5. Conclusion

In this paper, we have presented an anomaly detection scheme—called SCAN—that uses an enhanced EM clustering algorithm to cluster incoming network traffic. We evaluated SCAN's performance using the 1999 DARPA intrusion detection dataset. The salient features of SCAN include:

- The ability to detect anomalous network events with relatively high accuracy in the absence of complete audit data.
- Improvement in the clustering algorithm's convergence speed which was achieved by using a combination of data summaries, Bloom filters and associated arrays.

As part of our future work, we plan to investigate the feasibility of applying SCAN for real-time or near real-time anomaly detection. Another aspect of anomaly detection that we would like to explore is *network trend prediction*. We plan to study a dynamic Bayesian network based temporal detection paradigm that uses the collected data summaries for long term network trend prediction and rule adaptation.

<sup>8</sup>Accuracy of clustering is measured as the average percentage of correctly classified data over multiple runs.

## References

- [1] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, and Valdes. A "Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES)", Tech. Report SRI-CSL-95-06, Computer Science Laboratory. (1995)
- [2] B. Bloom "Space/Time Tradeoffs in Hash Coding with Allowable Errors." In Comm. of the ACM. (1970)
- [3] P. Bradley, U. Fayyad and C. Reina, "Scaling EM (Expectation-Maximization) Clustering to Large Databases", Microsoft Research Report MSR-TR-98-35. (1998)
- [4] A. Broder and M. Mitzenmacher, "Network Applications of Bloom filters: A Survey", In Proc. of Allerton Conference. (2002)
- [5] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm", J. Royal Statistical Soc. (1977)
- [6] E. Eskin, "Anomaly detection over noisy data using learned probability distributions", In Proc. of the Seventeenth Int. Conference on Machine Learning (ICML). (2000)
- [7] E. Eskin., A. Arnold, M. Prerau, L. Portnoy and S. Stolfo, "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data.", In D. Barbara and S. Jajodia (editors), Applications of Data Mining in Computer Security, Kluwer. (2002)
- [8] S. Guha, R. Rastogi, and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes", In Proc. of the 15th Int. Conference On Data Eng., Sydney, Australia. (1999)
- [9] E. M. Knorr, and R. T. Ng, "Algorithms for Mining Distance-Based Outliers in Large Datasets", In Proc. of the 24th Int. Conf on Very Large Databases, New York City, NY. (1998)
- [10] R. J. A. Little, and D. A. Rubin, "Statistical analysis with missing data", Publishers John Wiley and Sons, New York. (1987)
- [11] S. M. Lynch, "Missing Data Techniques", URL: <http://www.princeton.edu/~slynch/missingdata.pdf>
- [12] M. Mahoney, P. K. Chan, "PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic", Tech. Report CS-2001-2, Florida Inst. of Tech. (2001)
- [13] M. Mahoney, P. K. Chan, "Learning Models of Network Traffic for Detecting Novel Attacks", Tech. Report CS-2002-8, Florida Inst. of Tech. ((2002)
- [14] M. Mahoney, P. K. Chan, "Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks", Proc. SIGKDD, Edmonton, Alberta. (2002)
- [15] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse and other variants", Tech. Report, Dept. of Statistics, Univ. of Toronto. (1993)
- [16] C. Ordonez, and E. Omiecinski, "FREM: Fast and Robust EM Clustering for Large Data Sets", In Proc. of ACM CIKM. (2002)
- [17] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient Algorithms for Mining Outliers from Large Data Sets", In Proc. of the ACM Sigmod 2000 Int. Conf. on Management of Data, Dallas, TX. (2000)
- [18] B. Yocom., R. Birdsall and D. Poletti-Metzel, "Gigabit intrusion-detection systems", URL: <http://www.nwfusion.com/reviews/2002/1104rev.html>. (2002)
- [19] T. Zhang, R. Ramakrishna and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases", In Proc. of ACM SIGMOD. (1996)