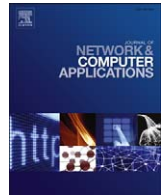




Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Critical infrastructure protection: Resource efficient sampling to improve detection of less frequent patterns in network traffic

Abdun Naser Mahmood^a, Jiankun Hu^{a,*}, Zahir Tari^a, Christopher Leckie^b

^a School of Computer Science and IT, RMIT University, Australia

^b Department of Computer Science and Software Engineering, Melbourne University, Australia

ARTICLE INFO

Article history:

Received 5 August 2009
Received in revised form
22 December 2009
Accepted 25 November 2010

Keywords:

Critical infrastructure
Traffic analysis
Sampling

ABSTRACT

Networked critical infrastructures are of national importance. However, such infrastructures are running 24/7. The supervisory control and data acquisition system (SCADA) of the critical infrastructure will generate enormous network traffic continuously. It is vital in such environments that only useful data are stored while redundant data are discarded to reduce the huge data storage demand. However it is technically challenging to reduce the demand on data storage while losing little information. In this paper, a resource conserving sampling technique is proposed to improve detection of less frequent patterns from huge network traffic under the fixed data storage capacity of the system. Such less frequent patterns are often related to subtle network intrusion activities. Experiments using the 1998 DARPA intrusion Detection Dataset have validated the effectiveness of the proposed scheme.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

Critical infrastructures, such as national power plants, are of national importance. For the operational performance, such critical infrastructures are often connected to the information network. There is a growing trend to utilize the public network to save communication cost. Accompanied by this is the increasing risk of network attacks or intrusions (Queiroz et al., 2008). New types of attacks are being generated daily, IDS and firewall technology cannot effectively block such new attacks (Hu et al., 2009; Islam et al., 2009; Xuan Dau et al., 2009). Therefore, there is a legal as well as security requirement to keep network traffic data for future forensic data analysis. However, critical infrastructures are often running 24/7. The supervisory control and data acquisition system (SCADA) of the critical infrastructure will generate enormous network traffic continuously. It is vital in such environments that only useful data are stored while redundant data are discarded to reduce the huge data storage demand. However, it is technically challenging to reduce the demand on data storage while losing little information. Such technology is often referred to as network traffic characterization. In addition to the need for network security, there is also another growing need to characterize network traffic data for a number of network management services (Mahmood et al., 2009). These include analysis of traffic volume, traffic dynamics, and traffic mixture.

Often these characterization tasks require using some form of data mining technique such as frequent itemset mining (Estan et al., 2003) or clustering (Mahoney and Chan, 2003; Duffield et al., 2001). However, most of these techniques perform poorly when large amounts of data need to be analyzed at or near network transmission speeds. Sampling is a popular technique for data reduction, and has been applied in various aspects of network management (Phaal et al., 2001), such as traffic measurement and reporting (Duffield and Grossglauser, 2000), traffic characterization (Claffy et al., 1993) and intrusion detection (Kodialam and Lakshman, 2003). In particular, an open problem in analyzing network traffic data is how to use sampling to reduce the volume of data to be analyzed so that rare patterns in the traffic can be recognized in an effective manner. In this paper, we present a two-stage adaptive sampling scheme to address this problem.

Traditional approaches to sampling are often inadequate to capture the underlying distribution of the data. In particular, for network traffic, sampling is influenced by the high volume of traffic from flash crowds and denial of service (DoS) attacks. When a DoS attack is active, a naïve sampling technique would be biased towards the distribution of the DoS traffic. When used with a clustering algorithm, a naïve sampling scheme would be unable to capture smaller clusters since they would be sampled less frequently than the DoS attack traffic. Our goal is to avoid wasting resources on clustering traffic records that have already been represented by large clusters.

Our approach is a two-stage sampling scheme. In the first stage, traffic is sampled at a higher rate relative to the overall sampling rate required by the user. This is followed by a selection stage where sampled traffic is systematically matched against a

* Corresponding author.

E-mail addresses: abdun.mahmood@rmit.edu.au (A.N. Mahmood), jiankun@rmit.edu.au (J. Hu), zahir.tari@rmit.edu.au (Z. Tari), caleckie@unimelb.edu.au (C. Leckie).

buffer of previously observed traffic patterns. Sampled traffic that matches a pattern in the buffer is then filtered through a second stage of sampling, so that only a subset of traffic of known patterns is passed to the clustering system. Most importantly, all traffic that did not match the buffer is passed to the clustering system as well.

A key advantage of this approach is that we can increase the proportion of computational resources that are spent on new or unusual traffic patterns. Note that we do not eliminate previously seen traffic patterns, but only sample them at a lower rate, thus making our sampling more efficient in terms of the number of different patterns captured. Consequently, this approach enables us to identify smaller but still significant clusters more accurately than if traditional sampling approaches were employed. The key contributions of this work are: (i) a novel two-stage sampling scheme for use in resource constrained traffic characterization, (ii) a strategy for selecting the sampling rates in each sampling stage, and (iii) an evaluation on a standard benchmark dataset which demonstrates that our scheme can achieve greater accuracy for smaller traffic clusters in comparison to traditional systematic sampling, without significantly degrading the identification of larger traffic patterns. This paper extends our previous work (Mahmood et al., 2006). In this paper, we added discussion on existing sampling algorithms and extended the discussion on sampling algorithms, highlighted some desired characteristics of the sampling techniques for network traffic data, included new experimental results comparing existing sampling techniques with our proposed technique.

The organization of the rest of this paper is as follows. Section 2 motivates the use of adaptive sampling in resource constrained environments where there is a need to identify smaller but potentially significant patterns and discusses related work. In particular, Section 2.1 discusses existing sampling algorithms and their limitations for use with network traffic data. Extending the discussion on sampling algorithms, Section 2.2 outlines some desired characteristics of the sampling techniques for network traffic data. Section 2.3 motivates the use of a non-uniform sampling technique with an example dataset. Section 3 formally describes the problem and establishes its scope. Section 4 describes our two-stage adaptive sampling scheme. Section 5 explains the objectives of the evaluation of our sampling technique. Section 6 describes the methodology and Section 7 discusses the various experimental results.

2. Motivation and related work: from elephants to mice

Due to the growth in the bandwidth of networks, the volume of network traffic data is often too large for analysis using traditional data mining techniques (Claffy et al., 1993). In particular, it is difficult to collect, store and analyze huge amounts of network traffic data. Consequently, there is increasing interest in scalable solutions for mining network traffic data. For example, system administrators need to identify significant categories of traffic that are consuming resources in the network, such as DoS attacks, flash crowd events or peer-to-peer traffic.

2.1. Discussion on existing sampling techniques

Sampling is a popular choice for reduction of input data in data mining techniques. There are many sampling methods in practice. We briefly describe a few of the popular sampling techniques.

2.1.1. Uniform random sampling

In uniform random sampling, the sample is drawn randomly from the population of objects using a uniform probability distribution (Hlawka, 1984), where each element of the population is equally likely to be chosen for the sample. In uniform random sampling each flow or packet is chosen according to this probability and each member of the population has an equal chance of being included in the sample (Cochran, 1977).

A variation in uniform random sampling selects samples based on the normal distribution of some characteristics of the elements, where the sampling probability of a value varies with the distance of the value from the mean.

Formally, if $X=(X_{in}$ et al., 2007), $i=1 \dots N$ is the sequence of elements to be sampled, and P_i is the probability that the element x_i will be selected for the sample then, for uniform random sampling

$$P_i = n/N, \quad \text{for } i = 1 \dots N$$

where $n \leq N$ is the *desired* sample size from the set of N elements. Note that the *actual* sample size is a random variable.

Fig. 1 shows the elements selected at random by the uniform random sampling algorithm. In network traffic the list of elements arrives in a certain order and the sampling algorithm needs to decide which element to sample. In this case, uniform random sampling picks elements with an equal probability. For example, it chooses the elements 3, 5, 8, and 10 that are represented by the vertical lines.

2.1.2. Stratified random sampling

In stratified random sampling, the population is considered to be divided into a number of categories or strata, where elements may be identified with their corresponding stratum. The sampling scheme selects a random element in each stratum, producing a stratified sample. The number of elements in each stratum of the sample is usually chosen to be proportional to the size of the strata in the population.

Formally, the population is divided into K strata of size N_k , $k=1, \dots, K$, such that $\sum N_k = N$. Each stratum k has its own desired sample size n_k from N_k . Thus an element x_i is sampled with probability

$$P_i = n_k/N_k, \quad \text{if } x_i \in \text{stratum } k$$

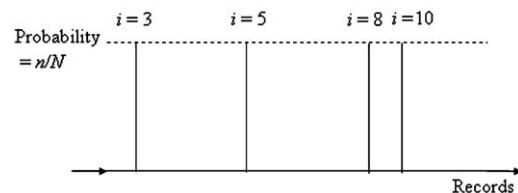


Fig. 1. Uniform random sampling with $k=4$. The vertical lines are the elements that are chosen with probability $P_i = n/N$. The figure shows elements drawn at the random positions 3, 5, 8 and 10 from the sequence of elements.

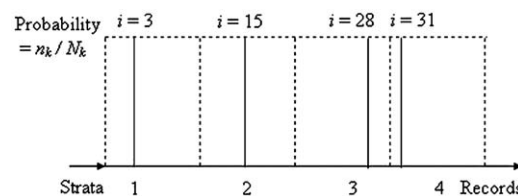


Fig. 2. Stratified random sampling with $K=4$ strata of equal size $N_k = N/K = 10$. The dotted vertical lines are the boundaries of the strata and the filled lines are the elements that are chosen with probability $P_i = n_k/N_k$. The figure shows elements drawn at the random positions 3, 15, 28 and 31 from the four strata.

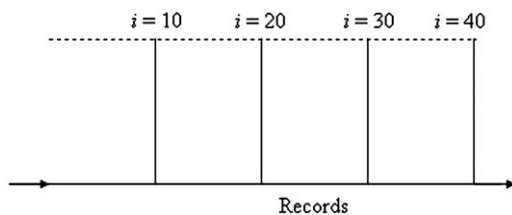


Fig. 3. Sampling probability of records in systematic sampling. Choosing records at positions 10, 20, 30, and 40 with an interval of $k=10$.

Fig. 2 shows the elements selected by stratified random sampling. The population is assumed to have distinct categories or strata. In this case there are four equal-sized strata and the sampling algorithm selects the elements from each strata based on the number of elements present in that stratum ($P_i=1/10$). This means that any element within a stratum is equally likely to be chosen by the sampling algorithm. However, the likelihood of choosing an element can vary across strata if required.

2.1.3. Systematic sampling

In systematic sampling, a record is sampled from the population, beginning from a given starting point to the end, at equal intervals. Each record in the sample is systematically picked from the population after a fixed number of elements have not been picked or after a fixed amount of time has passed since the last sampled element.

Formally, if $X=(X_i \text{ et al., } 2007)$, $i=1 \dots N$ is the sequence of elements to be sampled, and P_i is the probability that the element x_i will be selected for the sample then, for systematic sampling

$$P_i = \begin{cases} 1, & i \bmod k = 0, \\ 0, & i \bmod k \neq 0 \end{cases} \quad \text{where } k \text{ is the sampling period}$$

In other words, at any point in the sampling process the position of the next element to be sampled is predetermined. For example, in Fig. 3 a sample element is drawn from every 10th position sequentially from the population. The vertical line represents that the probability of choosing the particular element is 1, and the rest is 0.

Systematic sampling is easy to implement since it only involves choosing the element to be sampled at equal intervals. This is also an efficient sampling technique when either the population size or the sample size is known since the elements to be sampled can be found deterministically. However, systematic sampling may be vulnerable to periodicity in the population. If periodicity is present and the period is a multiple of k then sampling will be biased to the periodicity. One common method to reduce this bias is to choose the $k+i$ th element instead of the k th element, where i is chosen randomly between 1 and $(k-1)$.

2.2. Desired characteristics of our sampling algorithm

All three sampling algorithms discussed earlier have been widely used in different fields. The choice of sampling algorithm for a given application is highly dependent on the specific characteristics of the application domain. Although there are interesting theoretical results in the comparative parametric evaluation of the sampling algorithms (please see (Ehrenfeucht and Haussler, 1989; Bar-Yossef et al., 2001) for a rigorous discussion on the theoretical bounds on sample variance and size for different techniques), we are interested in the characteristics of a sampling algorithm that make them suitable to our network traffic domain. We identify three desirable characteristics of a sampling algorithm that can be used with network traffic data.

1. **Sequential sampling:** Traditional sampling techniques are designed for use on a fixed collection of data of known size. In contrast, sequential sampling is an approach (World Health Organization, 2006; Chromy, 1979) where the sample size is not fixed in advance since the total size of the population is not known. Instead, the available resources of the particular application dictate the choice of sample size. In the case of network traffic sampling, the sample size depends on available memory, processing power and network speed. In sequential sampling observations are collected individually or in small batches, and after each collection a decision is made from the accumulated data. Often there are not enough resources to retain all of the past data, therefore, this is sometimes also known as window based sampling in network stream data. Sequential sampling can be easily implemented with systematic sampling. As shown in Fig. 3, records arrive as a stream of flows and sampling is done on the records as they arrive. Due to the unknown size of the population and sample, random sampling is not directly applicable to sequential sampling.

2. **Non-uniform sampling:** Non-uniform sampling selects elements from the population based on some characteristics of the population or the desired sample. Uniform random sampling and systematic sampling are examples of uniform sampling, while stratified random sampling can be regarded as non-uniform sampling since it does not apply the same sampling probability across the strata. Next we discuss why uniform sampling, particularly uniform random sampling, is not suitable for network traffic applications.

Although sampling is a popular solution for data mining and statistical analysis of large datasets, such as network traffic flows, traditional sampling techniques are often inadequate to capture the underlying distributions (Claffy et al., 1993). For example, the density of the dataset heavily biases systematic sampling or uniform random sampling. Sampling a typical network trace containing a large number of probes or DoS attack packets will dwarf the other smaller attacks or interesting patterns. Uniform random sampling has been previously used in the context of clustering techniques (Ng and Han, 1994; Guha et al., 1998). Xu et al. (2005a, b) used random sampling to create a "Relative Uncertainty" profile of network flows, which is then used to characterize new flows using a frequent itemset mining technique similar to AutoFocus (Estan et al., 2003). The authors then use this technique to create a behavioral pattern, which can be used to block potential "exploit traffic" (Xu et al., 2005a, b) by constructing appropriate Access Control Lists for routers. Gonzalez and Paxson (2006), also inspired by AutoFocus, proposed packet level random sampling to detect heavy-hitters in network traffic.

It has been observed (Duffield et al., 2001; Feldmann et al., 2001) that a small number of heavy flows account for a large amount of traffic. Similarly, Estan and Varghese (2003) claimed that it is infeasible to accurately measure all flows on high speed links, but keeping track of only a few large flows (called "Elephants") may be sufficient. They proposed a *sample and hold* algorithm, which shares the common principle of counting samples with Gibbons and Matias (1998) but identifies large flows using less memory. In (Thompson, 1992; Thompson and Seber, 1996), sampling techniques are proposed for clustering based on the density of the clusters. Palmer and Faloutsos (2000) developed an algorithm to find clusters under the assumption of a Zipf distribution for the sizes of clusters. Kollios et al. (2001) proposed a variation of this technique using kernel based density estimation. Unlike these previous works, our emphasis is on the smaller flows that are often dominated by the larger flows. For example, in

uniform random sampling, every record has an equal probability of being sampled (Thompson, 1992). In the case of finding clusters for rare or infrequent classes of traffic, it is often desirable that rare records are sampled with a higher sampling rate than records belonging to frequent classes (Thompson and Seber, 1996).

In summary, the main focus of traffic analysis techniques such as those mentioned in (Islam et al., 2009; Estan et al., 2003; Cormode et al., 2003; Cormode et al., 2004) has been to focus on identifying the large traffic flows (Elephants), while ignoring the large number of smaller flows (Mice). However, many smaller clusters of flows can still contain relevant information. For example, many types of large traffic problems start out small, such as worm spread or distributed DoS attacks. The open issue that we address is how to develop a scalable data mining technique that can help to identify smaller traffic patterns in a computationally efficient manner.

The above discussion highlights the limitations of existing sampling algorithms in terms of addressing the problem of skewed distributions in network traffic data. Such skewness is present in many practical network traffic datasets, and we show that this occurs for the network traffic data generated for the DARPA intrusion detection datasets. In the next section, we show the composition of clusters in this intrusion detection dataset to highlight the importance of a sampling algorithm that can represent the smaller patterns present in network traffic data, such as certain types of network attacks.

Table 1

Top 10 attacks with their corresponding number of flows from days 1 to 25 of the 1998 DARPA traces.

Attack category	Flows	Attacks
DoS	1526628	Neptune
DoS	249609	Smurf
Probe	32632	Satan
Probe	15406	Ipsweep
Probe	10504	PortswEEP
DoS	10045	Pod
Probe	2356	Nmap
DoS	2172	Teardrop
DoS	1766	Warezclient
DoS	1281	Back

2.3. Motivating example with a labeled dataset

A labeled network traffic dataset contains flow records with labels identifying a particular class of behavior that is of interest. In network management, we are interested to identify the different types of user patterns and network conditions, for example, web browsing, P2P file transfer, and DoS attack. Therefore, it is useful to examine a packet trace with labeled patterns in order to demonstrate the relative distribution of sizes of the labeled traffic patterns. However, publicly available labeled traffic data are very rare because of security and privacy concerns (Mahoney and Chan, 2003). The Lincoln Laboratory DARPA intrusion detection data repository (Lincoln Lab, 1998) is one of the largest publicly available traffic traces. For example, the 1998 DARPA traffic traces include 25 days of traffic data with labeled attack information, collected from a purpose-built network, following the behavior model of both normal users and malicious users. In the absence of real life labeled data, these traces provide us with a classified set of traffic patterns, which can be used as the basis for evaluating methods to characterize network traffic.

It is interesting to note that although there may be many different types of attacks present in a dataset, the overwhelming majority of attacks (by the number of flows involved) are caused by Denial of Service attacks (*neptune* 8%, *smurf* 13%) and Probe attacks (*satan* 2%, *portsweep* 1%). The other attack types are User-to-Local (U2R) and Root-to-Local (R2L). More information about these categories and the attacks they represent can be found in Section 13.1 of Kendall's Master's Thesis on the DARPA Intrusion Detection Simulation Network (Kendall, 1999). Table 1 shows the top 10 attacks by volume in these traces and their categories. This gives us a picture of the relative frequency of different attacks in terms of the number of flows involved in the attack and their categories. Clearly, the most flow intensive attack categories are DoS and Probe. Any attempt to cluster flow records from traces such as these will be hampered by the volume of data to be analyzed. Moreover, if traditional sampling is used, most of the sampled records will still come from the top 5 to 10 traffic patterns as shown in Table 1.

Many of the smaller, but still significant patterns will be overlooked at low sampling rates. To illustrate this problem, consider the distribution of the sizes of clusters found as a result of clustering network traffic flows using the clustering technique described in (Cormode et al., 2004).

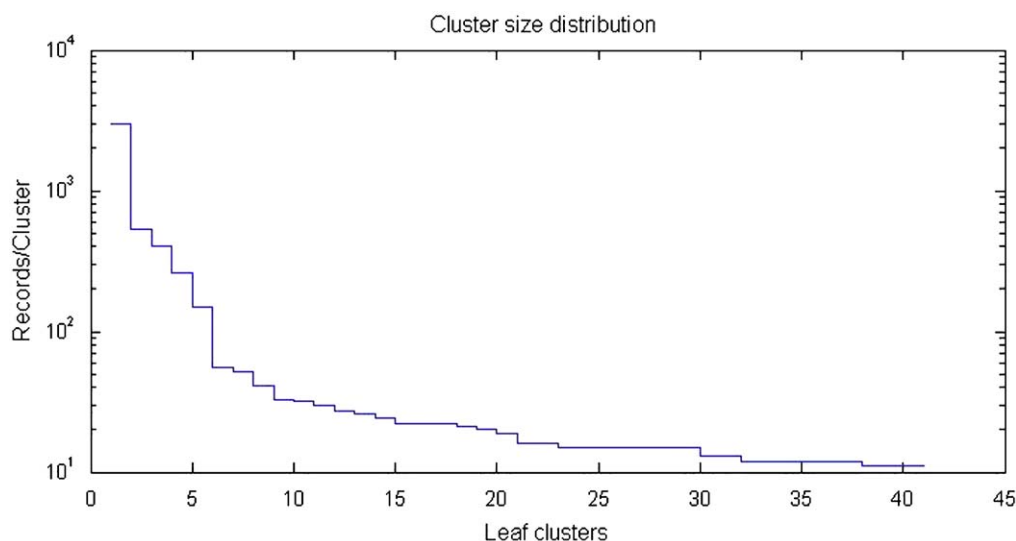


Fig. 4. Distribution of cluster sizes as a result of clustering dataset 6 (week 4, day 1) of the 1998 DARPA traces using the Echidna clustering tool.

We can see from Fig. 4 that there is a noticeable degree of separation between the sizes of the larger clusters and the sizes of the smaller clusters, indicating that the sizes of the clusters in the network traffic follow a heavy-tailed distribution. Thus, low rate sampling schemes will miss a significant proportion of the traffic patterns on this network. Our goal is to improve the scalability of clustering techniques for characterizing network traffic patterns, so that we have a better chance of identifying these smaller traffic patterns if our computational resources are constrained.

3. Problem statement

The scalability problem we address in this paper is relevant to any clustering algorithm, such as frequent itemset and partition-based clustering algorithms. However, we focus on with the scalability problem in the context of Echidna.

We are given as input a sequence of flow records that have been extracted from a network traffic stream, where each flow record is a 5-tuple $\langle \text{SrcIP}, \text{DstIP}, \text{Protocol}, \text{SrcPort}, \text{DstPort} \rangle$.

Our goal is to cluster these flow records into a set of generalized flows which represent significant patterns in the underlying traffic stream. Note that clustering is a resource constrained problem if the network traffic stream to be clustered is from a high speed network. Clustering is both CPU bounded, in terms of the maximum rate at which we can update clusters, and memory bounded, in terms of the maximum number of flows that can be kept in memory. In this context, if the arrival rate of new flow records exceeds the maximum rate at which we can update clusters with a new record, then we need to use some form of sampling in order to satisfy these resource constraints. Please see (Duffield et al., 2001) for more discussion on the necessity of sampling for analyzing high speed network traffic.

Traditional approaches to sampling network traffic include systematic, simple random, and stratified random sampling (Claffy et al., 1993). These approaches have the effect of sampling flow records from each underlying cluster at the same rate.

Suppose there are M classes of data among N records, where, $M < N$. For systematic sampling or random sampling, the probability of sampling an individual record for cluster C_m is

$$P_m = \frac{N_m}{N} S \quad (1)$$

where, N_m is the number of records from class C_m and S is the sampling rate, $0 < S \leq 1$.

In practice, however, the distribution of the sizes of the underlying clusters corresponding to the significant flows in the traffic is heavy-tailed (Duffield et al., 2001), i.e., many clusters contain only a small number of flows, while a few clusters contain many flows.

If traditional sampling techniques are applied in this case, then there will be a penalty in terms of the ability to accurately discover the smaller clusters. This is due to an observation by Guha et al. (1998) and Kollios et al. (2001), who noted that a minimum number of points (i.e., flows) need to be sampled from each cluster in order for those clusters to be recognized by the clustering algorithm. They assume that it is possible to identify a cluster in the population if at least a fraction f of the records in the cluster are present in the sample. This is acceptable for dense clusters in which the majority of the points lie close to the cluster centroid, so that a subset of the cluster can identify the cluster. In addition, if the separation between clusters is high compared to the intra-cluster distances, then even a small fraction f of the cluster is sufficient to distinguish between two clusters. Using Chernoff bounds (Motwani and Raghavan, 1995), Guha et al. (1998) show that for a cluster C_m containing N_m records, the error

in probability that the sampled cluster contains fewer than fN_m records is bounded in above by δ , where $0 \leq \delta \leq 1$, for the following condition on the sample size N_s :

$$N_s \geq fN + \frac{N}{N_m} \log\left(\frac{1}{\delta}\right) + \frac{N}{N_m} \sqrt{\left(\log\left(\frac{1}{\delta}\right)\right)^2 + 2fN_m \log\left(\frac{1}{\delta}\right)} \quad (2)$$

Consequently, if we are interested in the smallest cluster containing N_m records, then the sample must contain at least N_s records. Furthermore, if there are k clusters then the same sample with N_s records can represent the k clusters with probability of error at most $k\delta$.

Given this background, our aim is to increase the likelihood of finding a sufficient sample of records from small clusters, without needing to increase the overall sampling rate. Specifically, our goal is to develop a sampling scheme such that frequent clusters are sampled at a lower rate S_L while rare clusters are sampled at a higher rate $S_H > S_L$. In this way, we aim to increase the probability that a sufficient number of records are sampled from smaller clusters.

The key problem that needs to be addressed in this context is how to recognize, during sampling, whether an input record belongs to a rare or frequent cluster, given that the clusters are not all known a priori. In the next section, we describe the two-stage adaptive sampling scheme that we have proposed to address this problem.

4. Sampling architecture for network traffic clustering

The architecture of our two-stage sampling scheme is shown in Fig. 5. In the first stage, input records are sampled using uniform random sampling, with a sampling probability P_1 . Once a record has been selected as a result of the first sampling stage, we need to identify whether it belongs to one of the frequently occurring traffic patterns that have already been identified in the traffic trace. This is achieved by matching the record R against a buffer B of frequently occurring traffic patterns. If the record R does not match any frequent pattern in B , then R is considered as a new or less frequent pattern, and is passed directly to the clustering algorithm for inclusion into the clusters. In contrast, if R matches a frequent pattern in the buffer B , then R is considered to be less informative, since it already matches a known frequent pattern. Consequently, in that case, R would be passed to a second sampling stage where it would be sampled using uniform random sampling with sampling probability P_2 . If the record R passes this second stage of sampling, it is passed to the clusters, otherwise it is discarded. The effect of the second sampling stage is to reduce the rate at which known, frequent patterns are clustered, so that computational resources can be focused on characterizing the new or less frequent patterns.

There are two key research challenges in the design of this two-stage sampling scheme. The first challenge is how to select the sampling probabilities P_1 and P_2 given the constraint on the maximum throughput or overall sampling rate of the clustering algorithm, assigned by the user. The second challenge is how to populate and match entries in the buffer, which describe known frequent patterns in the traffic. We describe our approach to each of these problems in the subsections that follow.

4.1. Assignment of sampling probabilities

Given our two-stage sampling scheme as described above, we require a strategy for selecting the sampling probabilities P_1 and P_2 . Let SR denote the overall sampling ratio of the two-stage sampling scheme, i.e., the proportion of the original input records

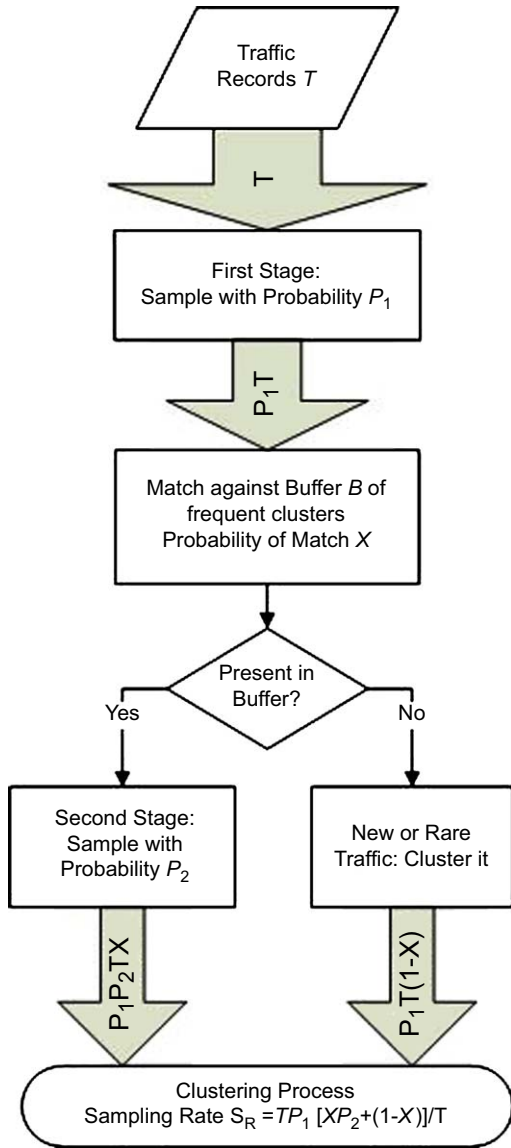


Fig. 5. Two-stage adaptive sampling scheme.

that are passed to the Echidna clustering subsystem. We can derive an expression for SR in terms of the initial sampling probability P_1 , the sampling probability of the second stage P_2 , and the proportion of records that match an entry in the buffer, which we denote as X

$$SR = P_1 [XP_2 + (1-X)] \quad (3)$$

Note that SR is the bound on the throughput of the clustering subsystem, and the match ratio X can be measured based on the observed frequency with which records match the buffer contents. Consequently, the two probability values P_1 and P_2 need to be further constrained.

In principle, if the match ratio X is large, then a large proportion of the traffic contains known, frequently occurring patterns. In that situation, we can afford to discard a large proportion of these recurring records, i.e., P_2 should be small. Conversely, if few records match the buffer, X is small, hence we need to retain a larger proportion of these records in order to refine these clusters, i.e., P_2 should be large. We can formulate this intuition using the constraint

$$P_2 = 1 - X \quad (4)$$

By substituting Eq. (4) into Eq. (3), we can derive an expression for P_1 as

$$P_1 = \frac{SR}{1-X^2} \quad (5)$$

Using this framework, we can control the importance of the match ratio X for patterns in the buffer by modifying the constraints in Eq. (4). If we wish to be more aggressive in filtering records that match the buffer, we can constrain P_2 as

$$P_2 = (1-X)^a, \quad a \geq 1$$

and derive P_1 accordingly. In that case, high values of the match ratio X would result in a lower sampling rate P_2 for the known, frequent patterns. Fig. 6 shows different options for setting P_2 using three different settings for the parameter a . Note that higher values of a result in the sample rate decreasing more rapidly as the match ratio X increases.

4.2. Management of the buffer of known frequent patterns

The role of the buffer in our two-stage sampling scheme is to provide a cache of known, frequent patterns in the network traffic. If a sampled input record R matches an entry B_i ($i=1, \dots, m$) in the buffer B , then it is of less interest to the clustering process. Key challenges in the design of the buffer are (1) how to represent entries in the buffer, and (2) how to populate these entries in the buffer. In Fig. 5, it was shown that a sample is tested against a buffer containing a list of frequent patterns. Such a list of patterns can be generated using a clustering algorithm from past data.

It is worth noting that the scalability problem we address in this paper is relevant to any clustering algorithm, such as frequent itemset and partition-based clustering algorithms. However, in this paper we focus on using a hierarchical clustering algorithm called Echidna (Mahmood et al., 2008), which we have previously developed for clustering network traffic data. In this section, we briefly describe the basic functionality of Echidna and how it generates frequent patterns. Please see (Mahmood et al., 2008; Mahmood et al., 2006) for a more detailed discussion and analysis of Echidna, and how it creates concise CF-entries.

Generating frequent patterns using Echidna clustering algorithm: Echidna is an agglomerative one-pass incremental hierarchical clustering algorithm that generates a cluster tree, also known as *Cluster Feature tree* or CF-tree. A CF entry represents a cluster of records in the form of a vector $\langle n, LS, SS \rangle$, where n is the number of records in the cluster, LS is the linear sum and SS is the square sum of the attributes of the records.

Echidna takes each record and iteratively builds a hierarchical tree of clusters called a CF-tree. In order to support these different types of attributes, Echidna provides an integrated approach to distance calculations, which incorporates distance functions for numerical, hierarchical and categorical attributes. In particular, Echidna can find clusters that represent a generalization of attribute values, such as a subnet that corresponds to a set of IP addresses. This provides a natural representation for describing a generalized pattern of network flows as a cluster.

Each cluster C_m is represented by a cluster feature vector that contains sufficient statistics to calculate the centroid \bar{c}_m and radius ρ_m of the cluster. Each data record R , corresponding to a 6-tuple traffic flow record $\langle SrcIP, DstIP, Protocol, SrcPort, DstPort, bytes \rangle$, is inserted by comparing R to the closest cluster starting from the root along a path P to a leaf node. At the leaf node, the data record R is inserted into the closest C_m and the radius ρ_m of the updated cluster is calculated. If $\rho_m > T$, where T is a threshold value in the range $[0,1]$, and if the number of CF-entries in the node is less than a minimum value, then R is inserted into the node as a new cluster. If a node has no more space for a new CF

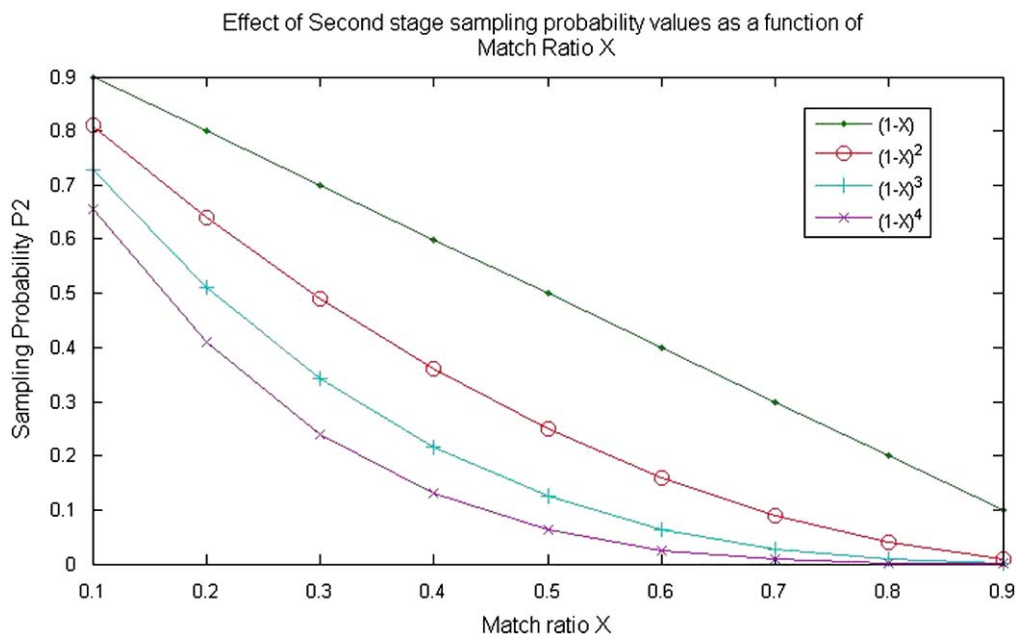


Fig. 6. Constraint on second stage sampling probability P_2 as a function of the match ratio X .

entry, then the node is split to create a new node and the path to the root is updated recursively.

Each leaf level cluster in the CF-tree corresponds to a generalized flow record, which may correspond to a source or destination IP sub-network, or a range of source or destination ports. The entries of the buffer correspond to leaf level nodes from the hierarchical cluster tree. Next, we describe how a buffer entry is created and how it is populated.

Buffer entry creation: First, the clustering algorithm builds a tree of clusters from incoming records. After the cluster tree has been built, the buffer creation phase begins. Because of the compact representation of the CF-entries, we use the CF-entries of the clusters as the entries in the buffer without any further modification.

Each entry is associated with two counters: the number of hits N_H and the last-hit counter N_{LH} . The N_H counter registers the number of times a record has matched this entry, which signifies the relevance of this pattern to the traffic and identifies the currently frequent patterns. However, the counter N_H does not enable us to test the temporal variation in the relevance of the corresponding pattern. The N_{LH} counter can be used to represent evolving patterns with more accuracy. For example, a buffer entry with a small value of N_{LH} has not been recently matched in the current trace. In contrast, a buffer entry with a large number of hits and a high value of the last-hit counter N_{LH} is more likely to have been relevant in the recent past of the traffic tree. We can use this knowledge of the state of the buffer entries in order to control the size of the buffer, identify old buffers entries that can be replaced with new entries, and implement more sophisticated buffer management functions. For example, we might want to dynamically lower the size of the buffer so that fewer entries need to be matched, in order to increase runtime efficiency of the sampling scheme. Alternatively, we may want to keep the buffer size fixed and replace those buffer entries which have been receiving less than $\lambda\mu$ hits for more than t periods with evolving buffers from the cluster tree, where λ is a fraction $[0..1]$, and μ is the average number of hits received by all buffer entries. These buffer management functions become important when the number of records in the population is unknown and when the clustering process is online and receiving *streaming* data, i.e., records that are constantly being input in a streaming fashion. For example, a

sampling scheme implemented in a network data analysis card (Anon, 2007) would benefit from a buffer management strategy since it continuously receives streaming data and there is a need to update the buffer as the patterns in the network traffic changes with time. As such, we have not used these functions in our implementation of sampling with the fixed size datasets.

Reasons for using packet header and not the packet content: As discussed above the buffer entries represent patterns of network traffic gleaned from the 6-tuple of packet header traces. The reason for choosing the packet header information only and not the packet content is as follows. There are two problems associated with trying to read packet contents for analysis. First, due to an increase in privacy and security concerns many protocols now support cryptographic measures to prevent man-in-the-middle attacks and unwanted interception of data over insecure media, thus making the packet payload unavailable for analysis. Second, even if the packet payload is available as plain text or decrypted for analysis, processing the payload is resource intensive and not scalable with the rate that packets arrive for medium to fast connections. Furthermore, existing network traffic monitoring for high speed networks also focus on techniques that deal only with packet headers. For example, Cormode et al. (2003, 2004) have used the 6-tuples from packet header traces and use an approximate count solution for a data stream environment. Kim et al. (2004) use the combination of rule-based packet/flow header detection and a traffic aggregation algorithm. Chhabra et al. (2005) proposed a randomized algorithm, which aggregates flows with similar field values of packet headers to yield signatures of network traffic. Finally, Estan et al. (2003) proposed a traffic clustering algorithm based on frequent itemset generation that also relies on packet headers only. Consequently, we limit our attention to packet headers traces which is often adequate to understand the broad level difference between attack and normal traffic.

Populating the buffer: From the leaf level, all clusters containing more than a threshold number of records T_r are inserted into the buffer. The choice of this “significance” threshold T_r is also used in report generation from the cluster tree, and is discussed in more detail in (Mahmood et al., 2008). After inserting the clusters into the buffer, the entries are sorted in non-decreasing order of the size of the clusters. This is done to save time in the linear search when matching

a record to a buffer entry. Since the larger clusters representing frequent patterns appear first, it is more likely for an incoming record to be matched at the top than at the bottom of the buffer.

We consider that a sampled input record R matches a buffer entry B_i if each attribute of R belongs to the range of values represented by the corresponding attribute in B_i . In order to find the similarity between R and B_i , the Euclidean distance metric distance (R, B_i) is used. As explained in (Mahmood et al., 2008), a traffic record is represented by a vector containing several types of variables or attributes. In the case of matching an IP address against a cluster entry, this only requires a match against the prefix of the two IP addresses. Similarly, in the case of a port this only requires testing whether both the ports are in the “low” or “high” ranges. Thus, the computational overhead of matching records against entries in the buffer is low and bounded by $O(d \times m \times P_1 \times \tau)$, where d is the number of dimensions of R (in our case $R=5$), m is the number of entries in the buffer, P_1 is the sampling rate of the first stage, and τ is the total traffic.

As described before, the entries in B are populated by extracting the clusters from the leaf level of the hierarchical cluster tree. While this is a straightforward process, we still have to address the issue of bootstrapping the system, given that the cluster tree needs to be generated before we can populate the buffer. We solve this by first clustering a sample of the traffic, without using the second sampling stage, i.e., this is equivalent to using our two-stage scheme with an empty buffer. The clustering process allows us to identify dominant flows in the traffic automatically. These dominant flows can then be extracted from the leaves of the cluster tree and used to populate the buffer B . In the current system, this is done once per packet trace. An issue for further research is how to incrementally update the buffer in non-stationary environments.

Recalling our observation in Fig. 4, there are a few big clusters containing a large proportion of the records compared to many small clusters containing small proportions of records. Thus, we expect that a relatively small buffer will be able to match a large proportion of the records sampled by the first stage of our scheme. This means that P_2 will be small, which reduces the proportion of computational resources that are applied to clustering known, frequent patterns in the data.

Hardware implementation issues: In order for our two-stage sampling scheme to be effective, the computational cost of sampling records and matching them against the buffer must be lower than the cost of insertion of a record into the cluster tree. As a result, it is preferable to de-couple the sampling phase from the data acquisition clustering phase of Echidna. In order to be able to lookup packets at line speed, our sampling technique can potentially be implemented inside a port of a router, such as a CISCO router, or a specialized network monitoring card, such as Ninja Probe Appliances (Endace Ninja Probe Appliances, 2007) from Endace (Anon, 2007). Since the creation of the buffer is independent of the sampling process, the sampling scheme can be constructed within a network card. Once the buffer has been filled with the clusters of interest either from a clustering algorithm or manually using rule-based entries, the network monitoring card can sample incoming records at line-speeds and match them against the contents of the buffer, which can be held on the router interface of the monitoring card. As shown in Section 6, a small buffer containing a few clusters is able to represent a large proportion of the input traffic records.

5. Objectives of evaluation

There is a trade-off between sampling and accuracy. In applications such as networking it is not possible to accumulate, as well as analyze, the gigabytes of network traffic data generated

every day. Instead, our aim is to filter out some of the repetitive patterns of flows while focusing on the less frequent patterns. In order to evaluate our two-stage sampling scheme, we have conducted an empirical evaluation in terms of (a) the effect of sampling on overall accuracy, (b) the effect of adaptive sampling on the detection of low volume traffic patterns, and (c) the computational efficiency of adaptive sampling compared to traditional systematic sampling.

As the basis for our evaluation, we required a dataset containing known traffic patterns. We have used the 1998 DARPA Intrusion Detection dataset (Lincoln Lab, 1998), as discussed in Section 2.3, to provide a set of packet traces containing known labeled patterns. From the dataset, 25 days of traffic traces from Week 3 to Week 7 were used in the experiments. As a basis for comparison with our two-stage adaptive sampling scheme we use systematic sampling (Cochran, 1977; Thompson, 1992; Thompson and Seber, 1996; Krishnaiah and Rao, 1988), which chooses records at an equal interval $I=SR \times N$ where SR is the sampling ratio and N is the total number of records. In the following subsections, all evaluations were performed on a time shared dual 2.8 GHz Xeon processor machine with 4 GB RAM running SunOS 5.9. The implementation of our algorithm was in Java version 1.5.

6. Evaluation methodology

While sampling saves computational resources, it has the potential to reduce the accuracy of clustering, since many examples or patterns are excluded in the learning process. In this section, we study the effect of reducing the sampling rate on the overall accuracy of our adaptive sampling scheme.

From the labeled traffic traces we identify the records as either belonging to an attack instance or as an instance of normal traffic. We measure accuracy by clustering the DARPA packet trace files, and measuring the number of sampled attack records that map into a cluster containing a majority of attack records. In this context, a true positive (TP) is an attack record in the trace that maps into a cluster containing a majority of attack records. A false negative (FN) is a known attack in the trace that, when inserted into the cluster tree, maps into a cluster that contains a majority of normal records. Similarly, a false positive (FP) is a normal record that maps into a majority attack cluster. The confusion matrix in Table 2 describes these parameters.

We can summarize the overall accuracy in terms of Precision and Recall as follows:

$$Precision = \frac{TP}{TP+FP}$$

reflects the number of true attacks detected by Echidna as a proportion of the total number of attacks reported. Similarly,

$$Recall = \frac{TP}{TP+FN}$$

Table 2
Standard confusion metrics for evaluation of traffic classification.

Actual connection label of record	Predicted connection label (majority class of matching cluster)	
	Normal	Attack
Normal	True negative (TN)	False positive (FP)
Attack	False negative (FN)	True positive (TP)

reflects the number of true attack records detected by Echidna as a proportion of the total number of attack records present in the dataset used for clustering.

7. Evaluation results

7.1. Evaluating accuracy: sensitivity to sampling rate

The effect on Precision and Recall of varying the sampling ratio SR in our two-stage adaptive sampling scheme is shown in Figs. 7 and 8. The results show the mean Precision and Recall values across the 25 trace files, along with error bars corresponding to ± 1 standard deviations. We show how Precision and Recall vary as the sampling ratio SR varies from 0.05 to 0.5. We can see that both Precision and Recall are around 80% and above. Moreover, there was no significant effect on the overall Precision and Recall as the sampling ratio was decreased. Thus, using our two-stage sampling scheme, we can achieve the same overall accuracy for lower SR settings using fewer computational resources, i.e., using a tenfold reduction in the sampling rate.

Since the threshold for selecting clusters as buffer entries was $T_i/\tau=0.1$, therefore there can be a maximum of 10 such clusters in the buffer, so the buffer size was set to 10. Note that although this was appropriate for the particular dataset we have used, the threshold value as well as the buffer size can be increased if even

smaller clusters are of interest. If the buffer size is increased then the buffer can accommodate more clusters and therefore represent smaller patterns as well as the larger ones. There is a trade-off between the resolution of patterns and the available memory. Since sampling rates are in the range of 1–10% many smaller patterns could potentially be lost, so a smaller threshold value for the buffer is recommended at the expense of using more memory. There are two techniques that can be used to find the size of the buffer. The first technique is calculating the mean and standard deviation of the clusters sizes at the leaf and choosing a value at the point where there are diminishing returns in increasing the buffer size. The second technique is dictated by the available memory, which fixes the size of the buffer. Note that in this technique the size of low frequency patterns represented in the buffer cannot be guaranteed since it is only known by looking at the leaf level clusters.

7.2. Comparison of sampling algorithms on accuracy

While the previous subsection demonstrates that there is no overall degradation in accuracy as a result of our adaptive sampling scheme, we also wanted to compare the performance of our algorithm compared to well-known sampling techniques, such as the systematic and random sampling algorithms mentioned in Section 2.1.

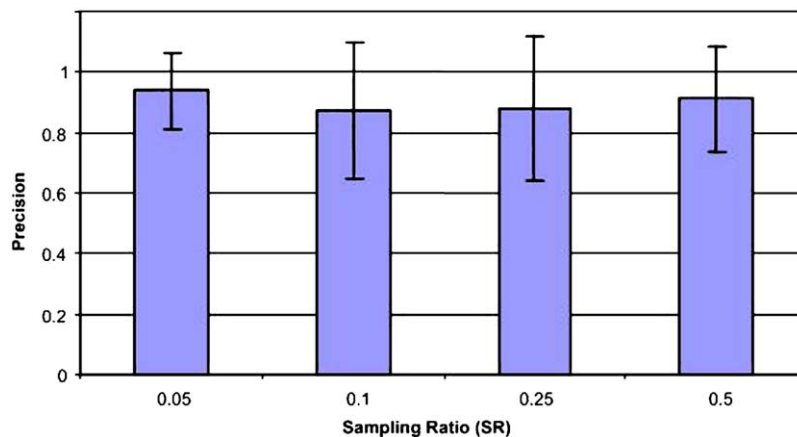


Fig. 7. Precision values of Echidna using adaptive sampling.

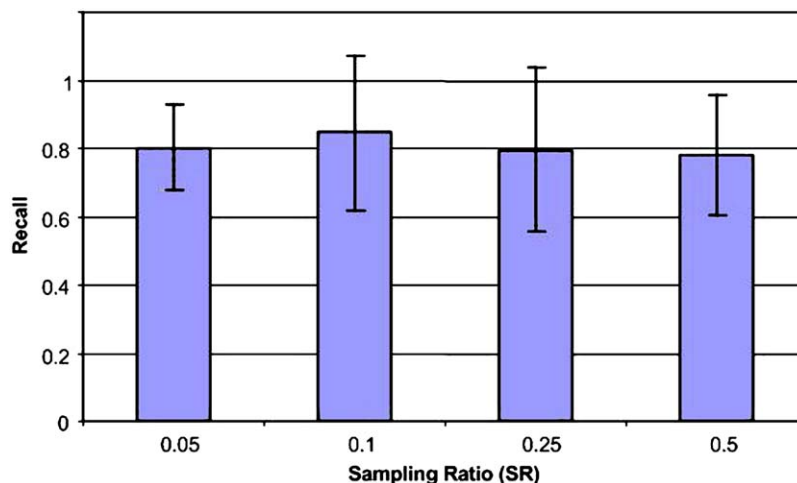


Fig. 8. Recall values of Echidna using adaptive sampling.

In Section 6, we mentioned about two related measures of accuracy, Precision and Recall. A measure that combines the results from both Precision and Recall is the F-measure (Rijsbergen, 1979), which is the weighted harmonic mean of Precision and Recall, as given below.

$$F_{measure} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Fig. 9 shows a comparison of accuracy for the popular sampling algorithms. The figure shows very clearly that for both the sampling rates (SR=0.12 and 0.25), adaptive sampling performs better than the traditional algorithms. The accuracy figures for systematic sampling and random sampling are very similar for both sampling rates.

7.3. Effect of adaptive sampling scheme on accuracy of detecting low frequency traffic patterns

In this experiment, we examine whether our scheme could improve the accuracy of clustering smaller flow patterns present in the packet traces. We analyzed the number of rare attack instances that were detected using our sampling scheme in comparison to systematic sampling. In particular

we studied the number of instances of these rare attacks that were detected as the sampling rate SR decreased from 0.25 to 0.12.

Figs. 10 and 11 show the number of rare attack instances detected using each sampling technique for sampling rates SR=0.25 and 0.12, respectively. We compare the number of different attacks detected using adaptive and systematic sampling for sampling rates 0.12 and 0.25.

In the case of SR=0.25, our two-stage adaptive sampling scheme performed as well or better than systematic sampling in 17 of the 18 attack types detected. In particular, there were three types of attacks (*land*, *loadmodule*, and *warez*) detected using our sampling scheme, which were not detected at all using systematic sampling. Moreover, as the sampling ratio SR decreased to 0.12, the difference in performance between two-stage adaptive sampling and systematic sampling was greater, with a larger number of attack instances detected by our scheme in comparison to systematic sampling. In both cases, adaptive sampling was able to detect some rare attack instances that were not detected by systematic sampling. For example, Fig. 11 shows that the attacks *eject*, *land*, *loadmodule*, *multihop*, and *warez* were detected using adaptive sampling, although they were not detected with systematic sampling.

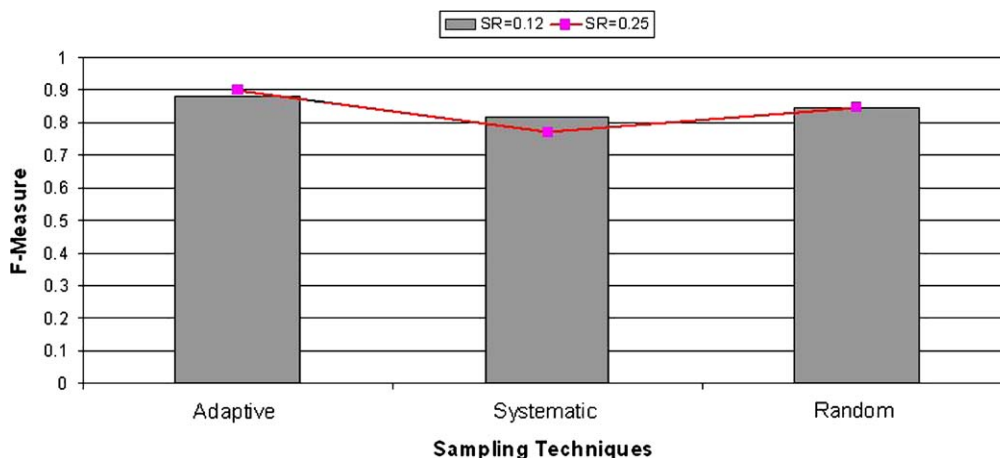


Fig. 9. Comparison of F-measure for sampling techniques.

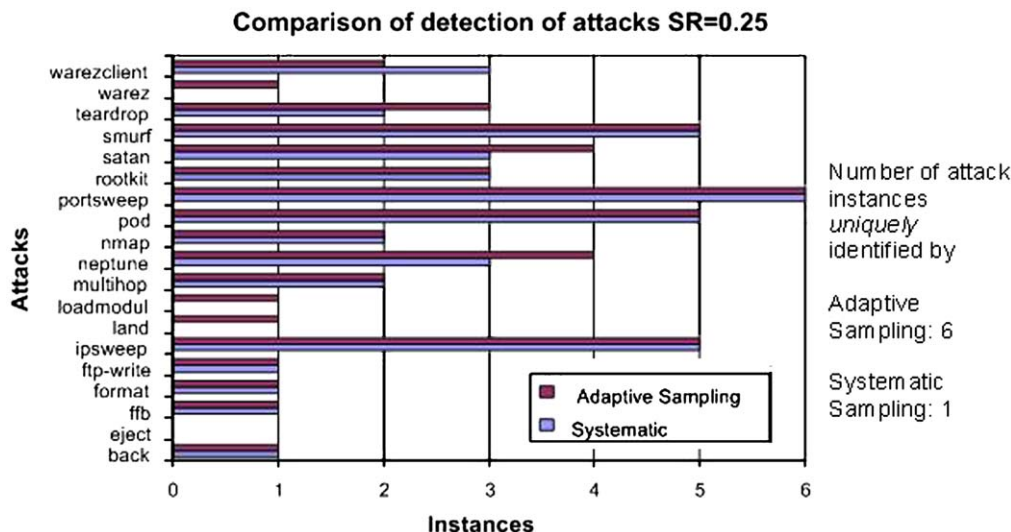


Fig. 10. Comparison of attack types detected using adaptive and systematic sampling for a sampling rate of SR=0.25.

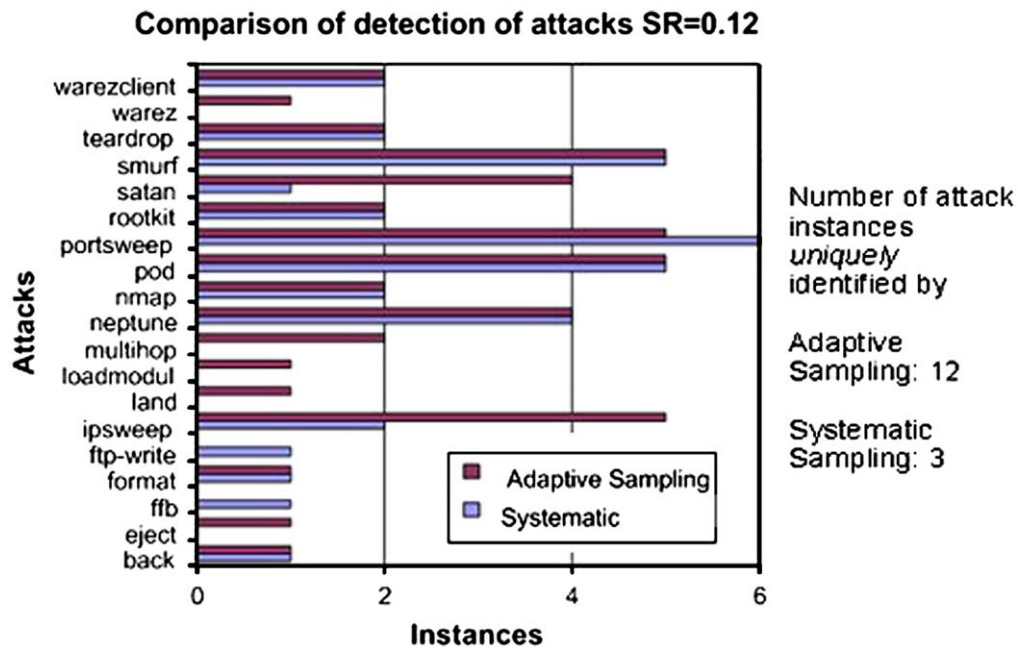


Fig. 11. Comparison of attack types detected using adaptive and systematic sampling for a sampling rate of SR=0.12.

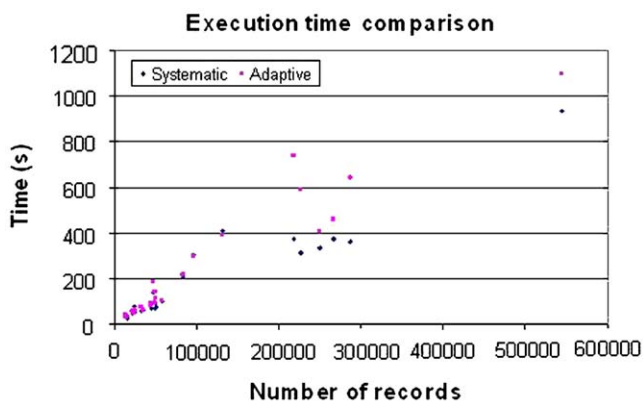


Fig. 12. Comparison of runtime between adaptive sampling and systematic sampling.

This provides evidence to demonstrate that at low sampling rates, our adaptive sampling scheme can achieve greater accuracy in detecting rare traffic patterns compared to traditional systematic sampling. This is consistent with our expectation that the adaptive scheme should achieve greater accuracy for rare patterns by diverting resources from known, high frequency traffic patterns.

7.4. Computational efficiency

The introduction of the buffer matching process adds additional computational overhead to the two-stage adaptive sampling scheme in comparison to systematic sampling. In this section, our aim is to measure the scale of this overhead. In order to measure this overhead, we have applied both our two-stage adaptive sampling and systematic sampling to packet trace files of different lengths. As a basis for comparison, both schemes have the same overall sampling rate, and the buffer is already populated with records. In this way, we can isolate the overhead

due to the buffer lookup process. Note that in practice the buffer can be constructed based on a sample of packet trace data.

The computation time required by each scheme is shown in Fig. 12. In all but six of the cases, the overhead caused by buffer matching is less than 30% and for 15 out of 25 instances the overhead is less than 20%.

Overall, both techniques scale linearly as the size of the packet trace to be clustered increases. Thus, there is only a small penalty for introducing the buffer into our sampling scheme, while it provides an advantage in terms of improving the detection accuracy of the clustering algorithm for less frequent traffic patterns. Since our current implementation of the buffer management scheme has not been optimized for efficiency, we expect that this overhead can be reduced further by using more efficient indexing schemes.

8. Summary

In this paper we have presented a scalable two-stage adaptive sampling scheme to characterize network traffic flows using a clustering algorithm. The sampling technique can be used with network traffic where common sampling techniques fail to identify smaller and rare traffic patterns due to a heavy-tailed distribution of pattern sizes. The main contributions of our approach are how to use and manage a buffer of known frequent patterns to prioritize sampled traffic, and how to adjust the sampling probabilities according to a user provided sampling rate.

We have shown experimentally using a standard benchmark dataset that the accuracy of the underlying clustering algorithm does not deteriorate significantly when the sampling rate is reduced using our sampling scheme. Furthermore, experiments with attack detection have demonstrated that our two-stage adaptive sampling scheme identified rare patterns and attacks that were not identified using systematic sampling. The runtime performance showed that execution time varies linearly with the number of records with a relatively small time overhead due to buffer matching. Adaptive sampling performed equally well for

small to medium sized datasets and showed a linear increase in time for larger datasets.

Our novel approach to adaptive sampling of network data raises several promising directions for future research. Interesting issues for further study include the effect of varying the buffer size on training time, as well as developing strategies for how to dynamically update the buffer as changes are observed in the underlying traffic patterns.

Acknowledgments

Partially supported by ARC Linkage Grant LP100100404 through Abdun Naser Mahmood, Jiankun Hu, Zahir Tari and the ARC Discovery Grant DP0985838, through Jiankun Hu.

References

- Anon. Endace network monitoring, latency measurement and application acceleration solutions. 2007 Available from; <<http://www.endace.com/>>.
- Bar-Yossef, Z, Kumar R, Sivakumar D. Sampling algorithms: lower bounds and applications. In: Proceedings of ACM symposium on theory of computing, 2001. p. 266–275.
- Chhabra P, John A, Saran H. PISA: automatic extraction of traffic signatures. In: Proceedings of Networking 2005, Berlin: Springer; 2005. p. 730–742.
- Chromy, J. Sequential sample selection methods. In: Proceedings of the American statistical association, survey research methods section, 1979. p. 401–406.
- Claffy K, Polyzos G, Braun H. Application of sampling methodologies to network traffic characterization. ACM SIGCOMM Computer Communication Review 1993;23(4):194–203.
- Cochran W. Sampling techniques. New York: 1977.
- Cormode G, Korn F, Muthukrishnan S, Srivastava D. Finding hierarchical heavy hitters in data streams. In: Proceedings of VLDB, 2003. p. 464–475.
- Cormode G, Korn F, Muthukrishnan S, Srivastava D. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. Proceedings of the 2004 ACM SIGMOD international conference on Management of data. New York, NY, USA: ACM Press; 2004. p. 155–166.
- Duffield N, Grossglauer M. Trajectory sampling for direct traffic observation. In: Proceedings of the conference on applications, technologies, architectures, and protocols for computer communication, New York, NY, USA: ACM Press; 2000. p. 271–282.
- Duffield N, Lund C, Thorup M. Charging from sampled network usage. In: Proceedings of the first ACM SIGCOMM workshop on internet measurement, New York, NY, USA: ACM Press; 2001. p. 245–256.
- Ehrenfeucht A, Haussler D. A general lower bound on the number of examples needed for learning. Information and Computation 1989;82(3):247–61.
- Endace Ninja Probe Appliances. 2007. Available from; <<http://www.endace.com/our-products/ninja-appliances/>>.
- Estan C, Savage S, Varghese G. Automatically inferring patterns of resource consumption in network traffic. In: Proceedings of the ACM SIGCOMM conference, 2003. p. 137–148.
- Estan C, Varghese G. New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice. ACM Transactions on Computer Systems 2003;21(3):270–313.
- Feldmann A, Greenberg A, Lund C, Reingold N, Rexford J, True F. Deriving traffic demands for operational IP networks: methodology and experience. IEEE/ACM Transactions on Networking 2001;9(3):265–79.
- Gibbons P, Matias Y. New sampling-based summary statistics for improving approximate query answers. In: Proceedings of 1998 ACM SIGMOD, New York, NY, USA: ACM Press; 1998. p. 331–342.
- Gonzalez J, Paxson V. Enhancing network intrusion detection with integrated sampling and filtering. Proceedings of RAID 2006:272–89.
- Guha S, Rastogi R, Shim K. CURE: an efficient clustering algorithm for large databases. In: Proceedings of 1998 ACM SIGMOD, New York, NY, USA: ACM Press; 1998. p. 73–84.
- Hlawka E. The theory of uniform distribution. AB Academic Publishers; 1984.
- Hu J, Qiu D, Chen HH, Yu X. A simple and efficient data processing scheme for HMM based anomaly intrusion detection. Special Issue of Advances on Network Intrusion Detection, IEEE Network 2009;23(1).
- Islam R, Zhou W, Xiang Y, Mahmood AN. Spam filtering for network traffic security on a multi-core environment. Concurrency and Computation: Practice and Experience 2009;21(10).
- Kendall K. A database of computer attacks for the evaluation of intrusion detection systems. MIT Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology; 1999 124 pp.
- Kim M, Kang H, Hung S, Chung S, Hong J. A flow-based method for abnormal network traffic detection. In: IEEE/IFIP network operations and management symposium, Seoul; 2004.
- Kodialam M, Lakshman T. Detecting network intrusions via sampling: a game theoretic approach. In: Proceedings of INFOCOM 2003, twenty-second annual joint conference of the IEEE Computer and Communications Societies, 2003.
- Kollios G, Gunopoulos D, Koudas N, Berchtold S. An efficient approximation scheme for data mining tasks. In: Proceedings of IEEE international conference on data engineering (ICDE'01), 2001. p. 453–462.
- Krishnaiah P, Rao C. Handbook of statistics 6: sampling. North-Holland; 1988.
- MIT Lincoln Lab. DARPA intrusion detection dataset. 1998. Available from; <http://www.ll.mit.edu/IST/ideval/data/1998/1998_data_index.htm>.
- Mahmood AN, Leckie C, Hu J, Tari Z, Atiq M. Network traffic analysis and SCADA security. In: Stamp M, editor. Handbook on information and communication security. Springer; 2009. p. 381–404.
- Mahmood AN, Leckie C, Udaya P. Echidna: efficient clustering of hierarchical data for network traffic analysis. Networking. Springer; 2006. p. 1092–1098.
- Mahmood AN, Leckie C, Udaya P. An efficient clustering scheme to exploit hierarchical data in network traffic analysis. Knowledge and Data Engineering, IEEE Transactions on 2008;20(6):752–67.
- Mahoney M, Chan P. An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection. In: Proceedings of RAID, Springer; 2003. p. 220–237.
- Motwani R, Raghavan P. Randomized algorithms. Cambridge University Press; 1995.
- Ng R, Han J. Efficient and effective clustering methods for spatial data mining. In: Proceedings of VLDB, 1994. p. 144–55.
- Palmer C, Faloutsos C. Density biased sampling: an improved method for data mining and clustering. In: Proceedings of 2000 ACM SIGMOD, New York, NY, USA: ACM Press; 2000. p. 82–92.
- Phaal P, Panchen S, McKee N. RFC 3176: InMon Corporation's sFlow: a method for monitoring traffic in switched and routed networks. Internet Engineering Task Force RFC 2001:3176.
- Queiroz C, Mahmood AN, Hu J, Tari Z, Yu X. Building a SCADA Security Testbed. In: Proceedings of the third international conference on Network & System Security (NSS09), Gold Coast, Australia: 2008.
- Rijsbergen CJV. Information retrieval. London: Butterworths; 1979.
- Thompson S. Sampling. New York: John Wiley and Sons; 1992.
- Thompson S, Seber G. Adaptive sampling. New York: John Wiley and Sons; 1996.
- World Health Organization. Sequential sampling. 2006.
- Xin, D., Han, J., Yan, X., Cheng, H. On compressing frequent patterns. Data and Knowledge Engineering 2007; 60(1):5–29.
- Xu K, Zhang Z, Bhattacharyya S. Profiling internet backbone traffic: behavior models and applications. In: Proceedings of the 2005 conference on applications, technologies, architectures, and protocols for computer communications, New York, NY, USA: ACM Press; 2005a. p. 169–180.
- Xu K, Zhang Z, Bhattacharyya S. Reducing unwanted traffic in a backbone network. In: Proceedings of steps to reducing unwanted traffic on the internet workshop (SRUTI), 2005b. p. 9–15.
- Xuan Dau H, Jiankun H, Peter B. A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference. Journal of Network and Computer Applications 2009;32(6):1219–228.