MINING NETFLOW RECORDS FOR HOST BEHAVIORS

by

Teja Kommineni

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

School of Computing
The University of Utah
May 2018

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Teja Kommineni**
has been approved by the following supervisory committee members:

Robert Ricci,	Chair(s)	**23 Feb 2017** Date Approved
Kobus van der Merwe,	Member	**23 Feb 2017** Date Approved
Suresh Venkatasubramanian,	Member	**23 Feb 2017** Date Approved

by **Ross Whitaker**, Chair/Dean of
the Department/College/School of **Computer Science**
and by **David Kieda**, Dean of The Graduate School.

ABSTRACT

Network administrators perform various activities every day in order to keep an organizations network healthy. They take the assistance of different tools to maintain these networks. The tools used by admins perform aggregation at different levels for performance management, security, maintaining the quality of service and others. Aggregating at port/application level to understand the behavior of flows is well studied whereas understanding the behavior of hosts and groups of hosts is less well studied. The latter in conjuction with the former is helpful to admins in making informed decisions regarding activities that include security policies and capacity planning among many others. Looking at flow level doesnt help us in understanding the total activities each host is undertaking and doesnt let us figure out which hosts are behaving alike as flow is just an instance of communication between two hosts. Hence, we want to aggregate by hosts and group them by understanding their behaviour.

As we have millions of flows and thousands of hosts to work on, we used data mining techniques to find the structure and present them to admins to help them make decisions and ease enterprise network management.

We have built a system that consumes flow records of network as input and determines the host behaviors in the network and groups the hosts accordingly. We also built an accompanying tool to this system that analyzes the host behaviors in different dimensions. This approach of extracting behaviors from network data has helped us in gaining interesting insights into the users of the system. We claim that analyzing host behaviors can help in uncovering the vulnerabilities in network security that are not found through traditional tools. We also claim that hosts behaving similarly require similar amount of network resources and this will be an efficient way for network admins to plan their network capacity compared to the present bandwidth monitoring techniques.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Understanding the behaviour of flows[1] at port level is well studied whereas understanding the behaviour of hosts and groups of hosts is less well studied. The latter in conjunction with the former is helpful to network admins in making informed decisions in day to day activities. Network administrators perform various network mangement activities and equipping them with right information will lead to better performance. Our system analyzes daily network data and extracts host[2] behaviors out of it. Behavior of a host is an aggregate of the amount of traffic that it sends both in terms of packets and bytes, the set of destinations it tries to connect to , the set of protocols it uses over some time period. For the purposes of our study the time period that we choose is a day. There is nothing inherent about choosing day as a time frame but we expect that the way networks are used varies from the days to nights and a day would be long enough to capture all these variations. These behaviors help us in learning more about what is going in the network.

Presently, admins use the following techniques to manage networks.

- They perform aggregation based on ports, protocols, applications. This helps in identifying applications consuming the most bandwidth, ports that are being used heavily, consolidate flow data from multiple devices, monitor wireless network bandwidth.

- They perform deep packet inspection to know what is in the payload of a packet. This is generally performed to extract the urls or sites that users are visiting. This is a fine grained version of the above technique.

---

[1] All packets with the same source/destination IP address, source/destination ports, protocol interface and class of service are grouped into a flow.

[2] A host in our system is defined as any unique IP address.

- Rule based systems are generally incorporated to secure the network from malicious users. In a rule based system a trigger is executed when specific actions occur. An example would be blocking the IP address of the user if he has more than 6 unsuccessful login attempts. Here the action is six unsuccessful attempts and the trigger is blocking the IP address.

Now, let us look at how does the present techniques differ from our proposed work. In the **Figure 1.1** on this page the left image represents a screenshot taken from Solarwinds[3] application when our data is passed through it. Here we can clearly see that the SolarWinds tool listed top 10 applications being used. If we look at the blue colored section of the pie chart it says that 83.67 percent of the traffic is World Wide Web traffic on port 80.

Moving on to the right side of the figure, it is a screenshot from the system we developed. Consider the hosts which are behaving as scanners on a Jan 3rd they were 1221 of them and on Jan 4th they are 1461 with an increase of 29 percent. One can clearly observe the difference between the two screenshots. While one views at the top applications and which ports make up the majority of traffic. The other views at the types of hosts and how they are changing over time. These are two different views of the same Network data.

[3]Solarwinds is an application that performs network monitoring using aggregation techniques when netflow data is passed as input.

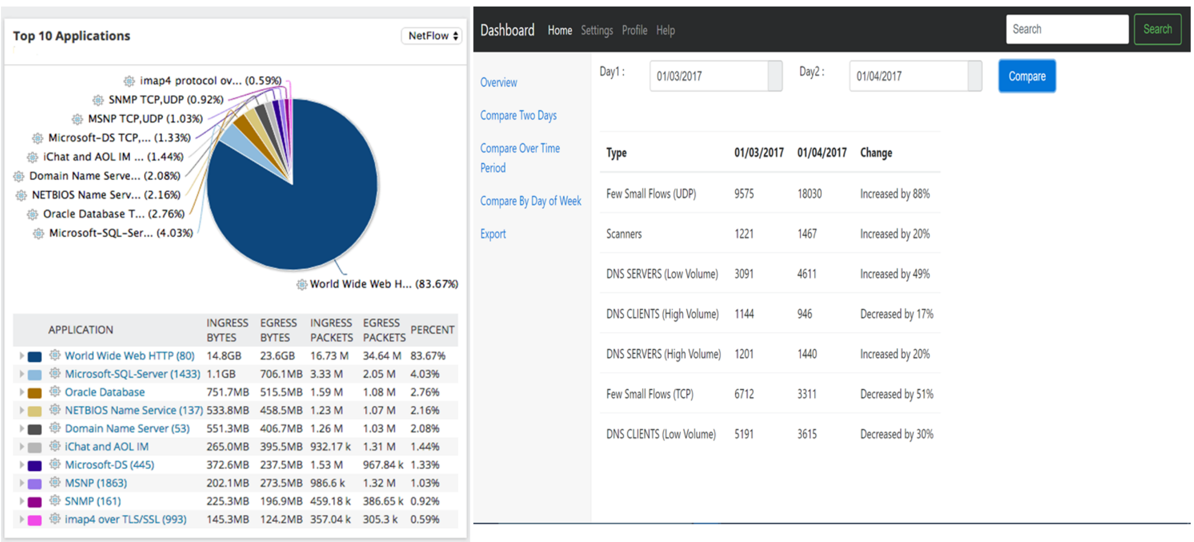


Figure 1.1. SolarWinds in comparison with our system

The obvious question that arises is why do we need an other view of network data and the answer is because the existing views are no more sufficient to perform different network management activites and our proposed approach helps in identifying behaviors that are not detected by traditional techniques. From the **Figure 1.1** on the previous page in the aggregation based techniques on the left side we are able to know that 86 percent of the traffic is web based. But what do we know about the hosts that are originators of these traffic. There could be hosts which are doing both DNS and WWW traffic. There could be hosts that are performing minimal amount of web traffic to evade from an anomaly event that they are performing. This information is not obtained from the SolarWinds or any other tool that uses these aggregation techniques. This is where we want our system to complement these existing techniques by providing hosts behaviors. Also, A major issue in network traffic analysis is classifying and characterizing traffic. Performing an accurate classification is indeed essential in various respects, such as traffic control, application identification, defense against attacks, and anomaly detection. However, they are often observed to fail, either because of packet encryption, arbitrary or dynamic port use, or because different protocols or utilizations employ the same single port. We also claim that hosts behaving similarly require similar amount of network resources and this will be an efficient way for network admins to plan their network capacity compared to the present bandwidth monitoring techniques. Some behaviors that could be of interest to the network admins are:

- If hosts are behaving as clients or servers. This gives admin a clear picture of how the network is being used and who are the major contributors to the traffic.
- If there are a group of hosts that exchange large number of packets containing small bytes then it is normal interactive traffic in the network.
- If a group of hosts are sending data to a single host. This could be an example of off-site back ups and network admin can use this information to plan the bandwidth accordingly.
- If a group of hosts are trying to scan on multiple ports then it is highly likely that these could be servers trying to infiltrate into the network and admins could take actions appropriately.

To extract these behaviors we have used data mining techniques. The reason for approaching these techniques is firstly the amount of data that we want to look into is huge. Secondly, we want to find behaviors/patterns a human may have not known to look in first place and data mining is a field of study that performs exactly the same task. It extracts patterns from a given data set and present it to the user in an understandable format. Specifically, we used an unsupervised approach called clustering to extract these behaviors. We also applied other data mining methods to measure the quality of our clustering and find the divergence of host behaviors over a given time.

1.1 Contributions

"By combining clustering and other data mining techniques we can produce a view of host behaviors that helps network administrators to understand behavior of the networks. This system complements existing network analysis tools to ease network management."

The following are the main contributions from this work which supports and provides evidence for the thesis statement.

1) We have built a system that uses data mining techniques to extract host behaviors from the given Netflow data.

2) We have built a tool to analyze the host behaviors in different dimensions and render it visually to help administrator take decisions.

The rest of this paper is structured as follows. Section 2 gives a background of data mining. Section 3 presents the design of our system. Section 4 provides the implementation details, while Section 5 discusses related work and plugs our work into context. Section 6 evaluates our system and demonstrates our claims. Finally, we outline items for future work in Section 7.

CHAPTER 2

BACKGROUND

Gaining insights from the network data using Machine Learning(ML) and Data Mining(DM) is a trending research area. There is a lot of confusion within the literature about the terms ML, DM as they often employ the same methods and therefore overlap significantly. Hence, below we describe the process of ML and DM briefly and establish why we have chosen Data Mining for building our system.

Machine Learning is a method of generating rules from past data to predict the future. A Machine Learning approach usually consists of two phases: training and testing. The following steps are usually performed:

- Identifying attributes (features) and classes from training data.
- Choosing a subset of the attributes for classification.
- Choosing a ML algorithm to create a model using the training data.
- Use the trained model to classify the unknown data.

Data Mining is the process of extracting patterns and structures implicit in the data and provide them in an understandable format to the user. Data mining is generally considered as a step in the process of Knowledge Discovery from Data (KDD). KDD usually consists of the following steps:

- Selection of raw data from which knowledge has to be extracted.
- Preprocessing the data to perform cleaning and filtering to avoid noise.
- Transforming the data so that all the attributes in the raw data are aligned towards achieving the common goal.
- Applying data mining algorithms to find rules or patterns.

- Interpret the observations of the above step and validate them.

As outlined above though both the techniques have similar steps of preprocessing data they differ on the end goal while ML maps the data set to known classes DM tries to find patterns out of the data set.

The decision of the approach that we want to employ in our problem solving also depends on the type of data we have in hand. If the data is completely labeled, the problem is called supervised learning and generally the task is to find a function or model that explains the data. The approaches such as curve fitting or machine-learning methods are used to model the data to the underlying problem. The label is generally a meaningful tag that is informative and has relation to the collected data. When a portion of the data is labeled during acquisition of the data or by human experts, the problem is called semi-supervised learning. The addition of the labeled data greatly helps to solve the problem. In general case the semi-supervised learning problem is converted to supervised learning problem by labeling the whole data set using the known labeled data and again the same machine-learning methods can be employed here. In unsupervised learning problems, we don't have any labels in the given data set and the main task is to find patterns, structures, or knowledge from this unlabeled data.

The dataset that we used for addressing our problem is flow data collected at routers which is explained in detail in the next section. This flow data generally comes without any labels. As, explained above if the data in hand doesn't have any labels the problem we are solving is called unsupervised learning problem. Also, in the introduction we have mentioned that the main goal of this work is to extract host behaviors from the aggregate data. This problem falls under a category of finding implicit/unseen patterns. Thus, having unlabeled data and the problem that we are trying to solve led us to use data mining techniques in building our system.

2.1 Unsupervised Learning

As we are approaching our problem using unsupervised techniques in data mining. Here is a brief explanation of different techniques within unsupervised learning. Unsupervised learning problems can be further grouped into clustering and association problems. A clustering problem is where you want to discover the inherent groupings in the data,and find patterns. An association rule learning problem is where you want to discover rules that describe large portions of your data, such as given an event X there is a chance of event Y happening. Since, we aim to find the inherent groupings our focus will be on clustering techniques. Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them.

Connectivity models [20], work on the premise that data points close to each other exhibit higher similarity than those lying far away. Cluster formation in these models can be done in two ways. Either we can initially consider all the points as a single cluster and then divide them into multiple clusters based on the distance between data points or we can assume each data point is a cluster on it's own and start merging them as the distance decreases. Hierarchical clustering algorithms fall under this category.

Centroid models, also use distance as their metric in grouping data points to cluster. While connectivity models consider distance between all points, centroid models measure distance of a point from cluster centers so as to assign it to the closest center. These models are iterative in nature and come to a halt when the cluster centers don't change. There are different techniques to initially choose cluster centers. K-Means clustering algorithm [8] is a popular algorithm that falls into this category. K-Center, K-medoids also belong to the same family.

Distribution models[15], Data sets can be fit into different distributions (eg: Normal, Gaussian). The clustering models that group all the data points with similar distribution and generate clusters according to these distributions fall under this category. A well known example of these models is Expectation-maximization algorithm.

Density Models [24], In these models clusters of data points are determined based on the variation of density in the data space. Different density regions with in the data space are found out and all those points that fall under same density region are assigned to the same cluster. Popular examples of density models are DBSCAN and OPTICS.

The choice of clustering model depends on the data in hand, amount of prior information that we have about the data and the problem we ought to solve. In our case we have chosen Centroid models to solve our problem and specifically the K-Means algorithm. Before, discussing about why K-Means in design section let us look at what K-Means algorithm is all about.

K-means [17] is a clustering algorithm that groups data points based on their feature values into K disjoint clusters. All the points that are classified into the same cluster are similar to each other when compared to those data points in the other clusters. The number of clusters K has to be specified in advance. Here are the four steps of the K-means clustering algorithm:

- Define the number of clusters K.
- Initialize the K cluster centers. We can arbitarily choose K data points from the given data set as cluster centers or apply K-means++ [1] algorithm to find the initial K cluster centers. K-means++ helps in careful initial seeding that can output better clusters.
- Iterate over all data points and compute the distances to the centers of all clusters. Assign each object to the cluster to which it is nearest.
- Recalculate the cluster centers after the above assignments.
- Repeat step 3 until the cluster centers do not change any more.

A distance function is required in order to compute the distance (i.e. similarity) between two objects. The most commonly used distance function is the Euclidean one which is defined as:

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

where $x = (x_1, ..., x_m)$ and $y = (y_1, ..., y_m)$ are two input vectors(data points) with m features. In the Euclidean distance function, each feature is weighted equally. However, in our data set different features are usually measured at different scales. Hence, they must be normalized before applying the distance function.

CHAPTER 3

DESIGN OVERVIEW

Detecting the behaviors implicit in the netflow data and using them in conjunction with existing tools to make network management easier is the overarching premise of this work. In this chapter we describe the design of our system, and how the different components fit into the architecture.

3.1 Components

Our design is comprised of the following essential components: Netflow Collection, Feature Engineering, Pattern Detector, and applications as shown in **Figure 3.1** on the following page.

3.1.1 Netflow Collection

While the term NetFlow has become a de-facto industry standard, many other network hardware manufacturers support alternative flow technologies namely Jflow, s-flow, NetStream, etc.. The University routers from which we have collected the data for this experiment have been equipped with NetFlow on it which led to using NetFlow as our flow technology.

3.1.1.1 What is NetFlow?

NetFlow is a proprietary protocol for collecting IP flow packets information on networks. There are different variants to this protocol but across all these versions a flow is defined uniquely by its source and destination IP addresses, source and destination ports, transportation protocol. Whenever one of these values changes a new flow is created. For each flow the number of packets, bytes and other network related information are tallied and stored in the routers cache till it expires, then this information is exported to a collector.

The captured flows are exported to NetFlow collectors at frequent intervals. The flows

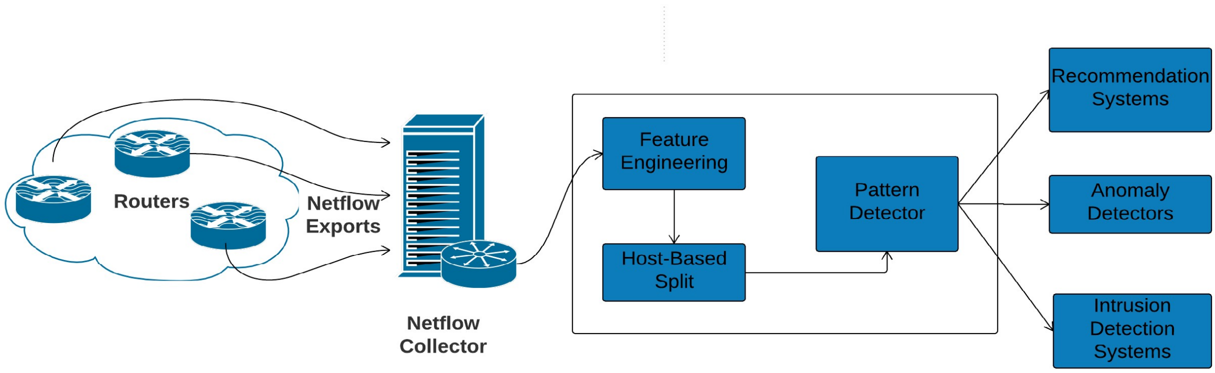


Figure 3.1. Architecture.

that are to be exported are determined based on the following rules: 1) When a flow is inactive for a certain time without receiving any new packets. 2) If the flow is active than max threshold time which is configurable. 3) If a TCP flag (FIN / RST) indicates the flow is terminated. NetFlow provides a powerful tool to keep track of what kind of traffic is going on the network, and are widely used for network monitoring. Most vendors support different flavors of similar flow monitoring approaches and a common standardization is done within the IETF IPFIX working group.

There are different variants of NetFlow with each advanced version giving additional information about the flow. The fields that we used in our experiment and that are supported across the versions are in **Figure 3.2** on the next page.

3.1.2 Feature Engineering

Feature Engineering is the process of transforming raw data into features so as to better represent the underlying structure of the data. This helps in increasing the accuracy of the models we build using data mining techniques. This step comes before modeling and after seeing the data. Features extracted from the data directly influence the models generated and predictions made out of it. The better the features are the better the results will be. The results that we achieve when we are using DM approaches are a factor of the data set

NetFlow Data – Flow statistics	
Source IP address	IP address of the device that transmitted the packet.
Destination IP address	IP address of the device that received the packet.
Source Port	Port used on the transmitting side to send this packet.
Destination Port	Port that received this packet on the destination device.
TCP Flags	Result of bitwise OR of TCP flags from all packets in the flow.
Bytes	Number of bytes associated with an IP Flow
Packets	Number of packets associated with an IP Flow

Figure 3.2. NetFlow Fields captured in a Flow

in hand, the features generated , the prediction model and the way the problem is framed. Most of the times even simpler models perform better if the features describe the structure inherent to the data. Below we describe different steps involved in the Feature Engineering with sample set of data.

3.1.2.1 Missing Data

Table 3.1 on the following page is a sample of network data collected using netflow. In total there are 40 columns (only 9 columns are shown in the table) for each record with each column capturing different flow information. From the table we can see that there are few missing values. Missing data is a common issue that every data science experiment has to deal with and there could be different reasons why netflow records could be missing some values such as, few filters could be turned on the router that doesn't let the netflow collector collect all the information, overloading of netflow collector and others. We handled Missing data using the following techniques

1) Replace missing values with the mean. For the features such as duration, total packets and bytes exchanged we assume that missing values are distributed similarly to the values that are present for each source IP. The rational approach here would be to substitute values using mean as it maintains the existing distribution.

Table 3.1. Netflow raw data.

Source IP	Destination IP	Source Port	Destination Port	Protocol	Duration	Flags	Packets	Bytes
59.2.154.56	155.98.47.116	11045	7547	TCP	318.0618	……	10	64516.125
223.157.2.51	155.98.44.63	7828	80	TCP			22	19800
190.72.57.37	155.98.47.29	44539	2323	UDP	318.0618	..s..	1	
223.157.2.51	155.98.44.63	24342	23		50.023	…..	50	45838.708
113.23.73.77	155.98.36.146	6176	80	TCP	318.0618	…..	17	
223.157.2.51	155.98.46.35	52336	23	TCP	12.6705	…..	50	388.098
184.105.247.2	155.98.34.233	44969	3389	UDP		……	50	19800
184.105.247.2	155.98.34.233	1318	21	TCP	40.230		121	19800
208.100.26.22	155.98.35.94	50861	53	ICMP	40.230		21	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

2) Replace missing values with the median/ mode. This is an other technique to handle missing data, It should be noted that using different statistical measures to replace missing values yield different solutions. Features that have categorical data cannot be handled using mean or median and hence we approach mode to fill in their missing values. The feature protocols was handled using the mode approach, filling the missing values with the most frequently occurring protocol value for that source IP.

3) The question that arises when replacing missing values is what percentage of values can a feature miss atmost. For example, filling in half the values of a feature in this step with a statistic measure is not reasonable. Features of this kind are considered as noise and could affect the effectiveness of the data mining algorithm. The better option here would be to discard the coulmn that has too may missing calues. In our case there were handful of columns that fell under this category and are discarded. flags shown in the **Table 3.1** on the current page is one among them. After handling missing values our data looks as in **Table 3.2** on the following page.

Table 3.2. NetFlow raw data after handling missing values

Source IP	Destination IP	Source Port	Destination Port	Protocol	Duration	Packets	Bytes
59.2.154.56	155.98.47.116	11045	7547	TCP	318.0618	10	64516.125
223.157.2.51	155.98.44.63	7828	80	TCP	**62.6935**	22	19800
190.72.57.37	155.98.47.29	44539	2323	UDP	318.0618	1	19800
223.157.2.51	155.98.44.63	24342	23	**TCP**	50.023	50	**20188.098**
113.23.73.77	155.98.36.146	6176	80	TCP	318.0618	17	388.098
223.157.2.51	155.98.46.35	52336	23	TCP	12.6705	50	388.098
184.105.247.2	155.98.34.233	44969	3389	UDP	**40.230**	50	**19800**
184.105.247.2	155.98.34.233	1318	21	TCP	40.230	121	19800
208.100.26.22	155.98.35.94	50861	53	ICMP	40.230	21	45838.708
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

3.1.2.2 Converting Categorical to Numerical Variables

Features could be either numerical or categorical variables. When dealing with algorithms that work on numerical data we have to make sure that the categorical variables are handled. For example, in our case feature protocol, has values such as TCP, UDP and others. One approach is to encode them as 0, 1, and 2 respectively converting them to numerical variables. This could pose a problem when using distance measuring algorithms as the distance between the encoded attributes, like 1 (UDP) ,0 (TCP) is smaller than 0 (TCP) ,2 (others) generating an unintended meaning to this feature and could lead to biasing of few values. Another approach would be to create a feature for each value of the categorical variable and mark it as 0 or 1 based on if the value is present or not. There are both pros and cons to this method. The pros are if we choose to aggregate the values on all features this conversion gives a numerical value to work with. The cons are scaling and standardization could be affected. **Table 3.3** on the next page represents data set after this step.

Above two steps fall under a family of techniques called data cleaning. After the data cleaning and removing the irrelevant columns of data using domain knowledge we aggregated netflow data based on source IP as our work focuses learning about hosts from their aggregate data. The **Table 3.4** on the following page shows a sample of aggregated

Table 3.3. After Converting categorical data to numerical

Source IP	Destination IP	Source Port	Destination Port	TCP	UDP	ICMP	Duration	Packets	Bytes
59.2.154.56	155.98.47.116	11045	7547	1	0	0	318.0618	10	64516.125
223.157.2.51	155.98.44.63	7828	80	1	0	0	62.6935	22	19800
190.72.57.37	155.98.47.29	44539	2323	0	1	0	318.0618	1	19800
223.157.2.51	155.98.44.63	24342	23	1	0	0	50.023	50	20188.098
113.23.73.77	155.98.36.146	6176	80	1	0	0	318.0618	17	388.098
223.157.2.51	155.98.46.35	52336	23	1	0	0	12.6705	50	388.098
184.105.247.2	155.98.34.233	44969	3389	0	1	0	40.230	50	19800
184.105.247.2	155.98.34.233	1318	21	1	0	0	40.230	121	19800
208.100.26.22	155.98.35.94	50861	53	0	0	1	40.230	21	45838.708

data with ten features. The first record in the **Table 3.4** on the current page conveys the information that in the given data set the host 59.2.154.56 (source IP) had appeared 450 times. Out of those 450 instances it had contacted 116 unique hosts. A total of 110 different source ports were used in the 450 flows and all the packets were sent to single destination port. The columns TCP, UDP, ICMP indicate how many of these flows fall under each category. So of the 450 flows there are 406 flows in which TCP protocol was used , 4 had UDP protocol used and the rest 40 had used ICMP protocol. And the columns Total Duration, Total Bytes, Total Packets indicate the respective information exchanged by this sourceIP on a whole. This aggregated data is what we use for learning host behaviors. But, before that we have to apply few other feature engineering techniques as described below.

Table 3.4. Aggregated data by Source IP

Source IP	Total Flows	Unique Destinations	Unique Source Ports	Unique Destination Ports	#TCP	#UDP	#ICMP	Total Duration	Total Packets	Total Bytes
59.2.154.56	450	116	110	1	406	4	40	318.0618	10	64516.125
223.157.2.51	3	2	78	80	3	0	0	62.6935	22	19800
190.72.57.37	84	10	10	2	70	14	0	318.0618	1	19800
113.23.73.77	18	18	1	23	18	0	0	50.023	50	20188.098

3.1.2.3 Feature Scaling

Feature Scaling/Feature Normalization, a method used to standardize the range of independent features of data. Failure to standardize variables might result in algorithms placing undue significance to variables that are on a higher scale. For example from the **Table 3.4** on the preceding page it is evident that the total packets and total bytes have higher magnitude compared to total flows and destinations. This range difference shouldn't bias our results. There are different ways to do feature scaling namely, Min-Max scaling, Variance scaling, L2 normalization etc.. The choice depends on the data and different statistics of it such as Mean, Variance, L2 norm. Scaling is not advisable in all instances we might loose valuable information if applied without proper thought. In our case we have chosen simple Min-Max scaling as our goal was merely to change the range of the data and not the distribution and Min-Max scaling helps us in achieving this at a lower cost.

3.1.2.4 Feature Transformation

Transforming Non-normal distribution to Normal, many ML/DM tools perform well on normalized data and having skewed data will give inaccurate results. Log transformations are generally used to normalize the skewed data which is the case with most network data. After transforming the data if it doesn't capture the essence of original data we could look at other options such as square root, cube root. BoxCox is also another technique that changes the distribution of variables from non-normal to normal or near normal. **Figure 3.3** on the next page shows the distribution of a feature Total Flows from our aggregated data, as seen from the graph it is heavily skewed towards left and this can be directly attributed to the data set we have in hand. The graph indicates that in most instances number of flows in which a source IP appears is less than 100 and hence we employ transformation techniques to decrease the amount of skewness in the data. **Figure 3.4** on the following page shows the same data after applying log transformation.

3.1.2.5 Feature Selection

Feature Selection refers to the process of selecting a subset of relevant features to model the data. It helps in shorter time periods of execution, overcoming the overfitting problem and making the solution more generic and avoid curse of dimensionality. The central

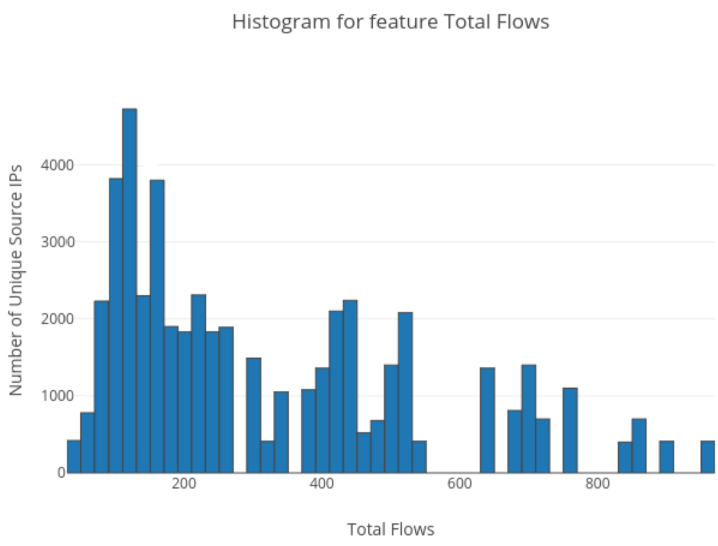


Figure 3.3. Skewed data before transformation

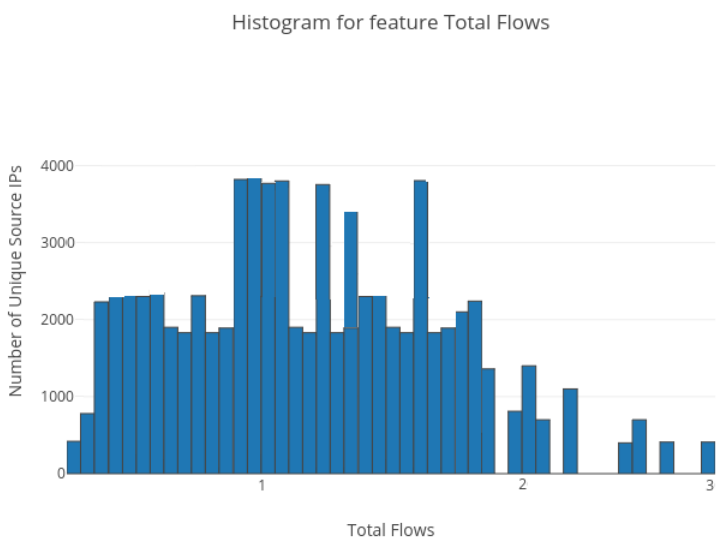


Figure 3.4. Normalized data

premise of this process is to remove redundant or irrelevant features that don't make much sense to the problem we ought to solve. Feature selection and dimensionality reduction are two confusing terms. Though, both methods seek to reduce the number of attributes in the dataset, but dimensionality reduction does it by creating new combinations of attributes, where as feature selection methods include and exclude attributes present in the data without changing them. Principal Component Analysis, Singular Value Decomposition are examples of dimensionality reduction methods. The techniques used to do feature selection depend on text, numerical variables and they are three general classes of them Filter, Wrapper, Embedded methods. Feature selection is a much tougher problem in unsupervised setting as we don't have any labels that helpus in determining the information gain with and without a feature.Problems of this kind have been rarely studied in the literature [5] [7].

The common strategy of most approaches is the use of filter methods Filter model methods do not utilize any clustering algorithm to test the quality of the features. They evaluate the score of each feature according to criteria [7] mentioned here. It selects the features with the highest score. It is called the filter since it filters out the irrelevant features using given criteria. Wrapper methods are an other class of methods used for feature selection where we train the given model with different combinations of features and make inferences accordingly. Some common examples of wrapper methods are forward feature selection, backward feature elimination and recursive feature elimination. Forward selection is similar to elbow method explained in the next section. It is an iterative technique where we start with zero features and in each step we keep adding feature that has the most information gain or best expresses the underlying structure. This procedure is stopped when addition of an extra feature doesn't add any significant information to the model. forward The backward eliminaion technique is exactly opposite to the Forward selection method. Here we start with all the features and remove the least significant feature at each iteration. This iterative procedure stops when no improvement is observed on removal of features.

We have chosen wrapper methods backward elimination technique for feature selection as it measures the usefulness of a subset of features by actually training a model on it. Though, it is computationally expensive compared to Filter based approaches in our context it was reasonable as we had only 10 features to look into. The result of feature selection is we have detected two features that have very less information gain namely the ICMP ports and duration and hence we removed these two from our feature list making our final feature count to 8. Feature Selection has helped us in enabling the machine learning algorithm to train faster. It reduced the complexity of the model and made it easier to interpret. We were also able to address the overfitting problem.

3.2 Pattern Detector

Pattern Detector consists of the core logic of our system. After cleaning the data and applying feature engineering we send the data through a pattern detector. Pattern Detector sends the data through a clustering algorithm. It then maps the generated clusters to the host behaviors which is explained later.

As mentioned in the background section we choose to address our problem using unsupervised learning approaches. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

3.2.0.1 Why K-Means ??

In the section 2.1 we have discussed about different type of clustering techniques in detail. Here we provide a reasoning for choosing K-Means.

Connectivity models like Hierarchical clustering don't have a provision for relocating objects that may have been incorrectly grouped at an early stage. Also, use of different distance metrics for measuring distances between clusters may generate different results. Performing multiple experiments and comparing the results is recommended to support the veracity of the original results. Lastly, time complexity of at least O($n^2 logn$) is required, where n is the number of data points thus lacking scalability for handling big datasets.

Distribution Models assume a distribution for data but for many real data sets, there may be no concisely defined mathematical model (e.g. assuming Gaussian distributions is a rather strong assumption on the data). Also, though distribution models have an excellent theoretical foundation but they suffer from one key problem known as overfitting.

We have experimented our data set with the density model DBSCAN and we have found the data being grouped at most in to only two clusters. The reason for this is that density based models expect some kind of density drop to detect cluster borders. Recall from the above section that our data set has many source IPs that appear in less than 100 flows and hence there wasn't any drop in density observed resulting in bad clustering.

Experimenting with centroid models especially K-Means gave us an advantage to play with the variable K and view into how host behaviors are being grouped with different values of K. It is scalable for heavier datasets and doesn't suffer from the overfitting problem. Also, there isn't any inherent assumption on which distribution the dataset should follow. Thus, making it a clear choice for building our system.

3.2.0.2 How to choose K ??

In many situations, parameters and variables chosen by the algorithm or by users affects the performance of the algorithm differently and results in different outputs. Similarly, an essential problem of the K-means clustering method is to define an appropriate number of clusters K. In the literature there are different approaches proposed to determine an accurate value of K. These methods include direct methods and statistical testing methods. Direct methods consists of optimizing a criterion, such as the within cluster sums of squares or the average silhouette. The corresponding methods are named elbow and silhouette methods, respectively. Statistical testing methods consists of comparing evidence against null hypothesis. An example is the gap statistic. We have used the direct methods as they optimize a criteria the elbow method to determine K and then cross-verified the values with the Silhouette Index obtained for this same data set.

Recall that, the basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the total intra-cluster variation [or total within-cluster sum of square (WSS)] is minimized. The total WSS measures the compactness of the clustering and we want it to be as small as possible. The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesnt improve much better the total WSS. The optimal number of clusters can be defined as follow:

- Compute k-means clustering for different values of k. For instance, by varying k from 1 to 50 clusters.
- For each k, calculate the total within-cluster sum of square (W_k).

$$D_k = \sum_{x_i \in C_k} \sum_{x_j \in C_k} ||x_i - x_j||^2$$
$$W_k = \sum_1^K \frac{1}{n_k} D_k$$

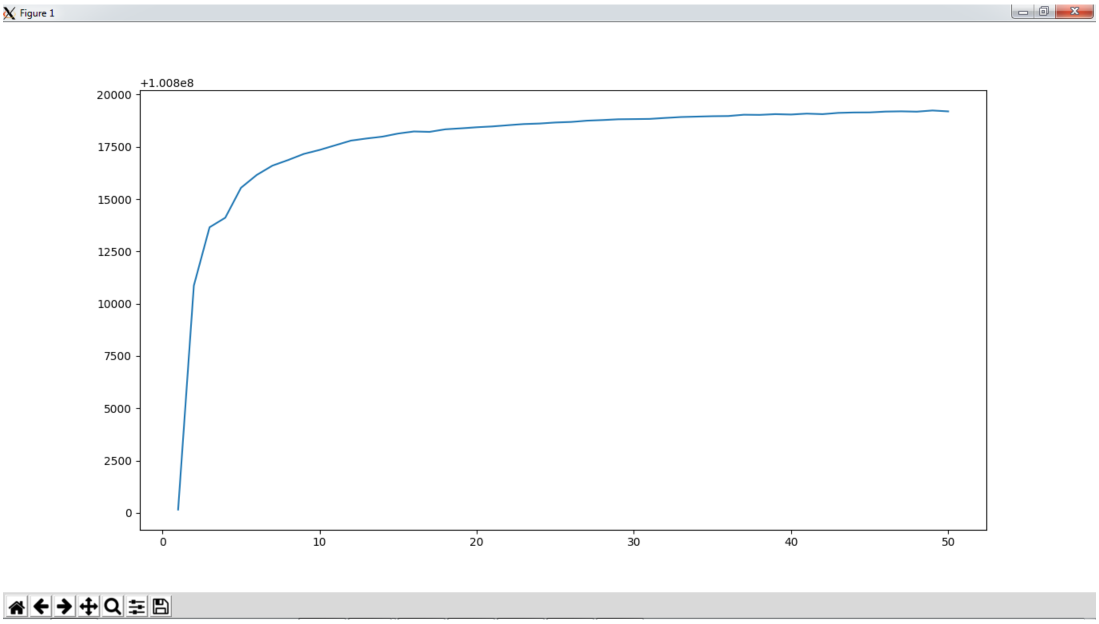


Figure 3.5. Elbow Method

Where K is the number of clusters, n_k is the number of points in cluster k and D_k is the sum of distances between all points in the cluster.

- Plot the curve of wss according to the number of clusters k.

- The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

The graph depicts the amount of information we are gaining with addition of each cluster. the first clusters will add much information, but at some point the marginal gain will drop, giving an angle in the graph. The number of clusters are chosen at this point, hence the elbow criterion. **Figure 3.5** on the current page shows the elbow curve obtained for our data set on a chosen day. This method gave us a value of 7 for K.

However, the elbow method is not unambiguous. especially if the data is not very clustered we will notice that the elbow chart will not have a clear elbow. Instead, we see a fairly smooth curve, and it's unclear what is the best value of k to choose. In cases like this, we might try a different method for determining the optimal k. Hence, to corroborate the claim of elbow method for K we also ran our data set through another technique called silhouette.

The average silhouette approach measures the quality of a clustering. That is, it determines how well each object lies within its cluster. A high average silhouette width indicates a good clustering. Average silhouette method computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximize the average silhouette over a range of possible values for k. The algorithm is similar to the elbow method and can be computed as follow:

- Compute k-means clustering for different values of k. For instance, by varying k from 1 to 50 clusters.
- For each k, calculate the average silhouette of observations (avg.sil).

  $$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

  Here s(i) is the silhouette index of a cluster point i. Where a(i) is the average distance of point i to all the objects in the same cluster while b(i) is the minimum average distance from the point i to all the points of different cluster. avg.sil is calculated by finding mean of all the s(i) for each point in a cluster.
- Plot the curve of avg.sil according to the number of clusters k.
- The location of the maximum is considered as the appropriate number of clusters.

The intuition behind this approach is for a point as a(i) is a measure of how dissimilar i is to its own cluster, a small value means it is well matched. Furthermore, a large b(i) implies that i is badly matched to its neighboring cluster. Thus an s(i) close to one means that the data is appropriately clustered. if s(i) is close to negative one, then by the same logic we see that i would be more appropriate if it was clustered in its neighboring cluster. The average s(i) over all data of a cluster is a measure of how tightly grouped all the data in the cluster are. Thus the average s(i) over all data of the entire dataset is a measure of how appropriately the data have been clustered. **Figure 3.6** on the following page shows the the plot obtained using silhouette technique.

3.2.0.3 Cluster Labeling and Comparison

The output of the pattern detection step is a set of clusters. We explored these clusters to better understand the grouping of hosts. We observed that each cluster exhibits a unique

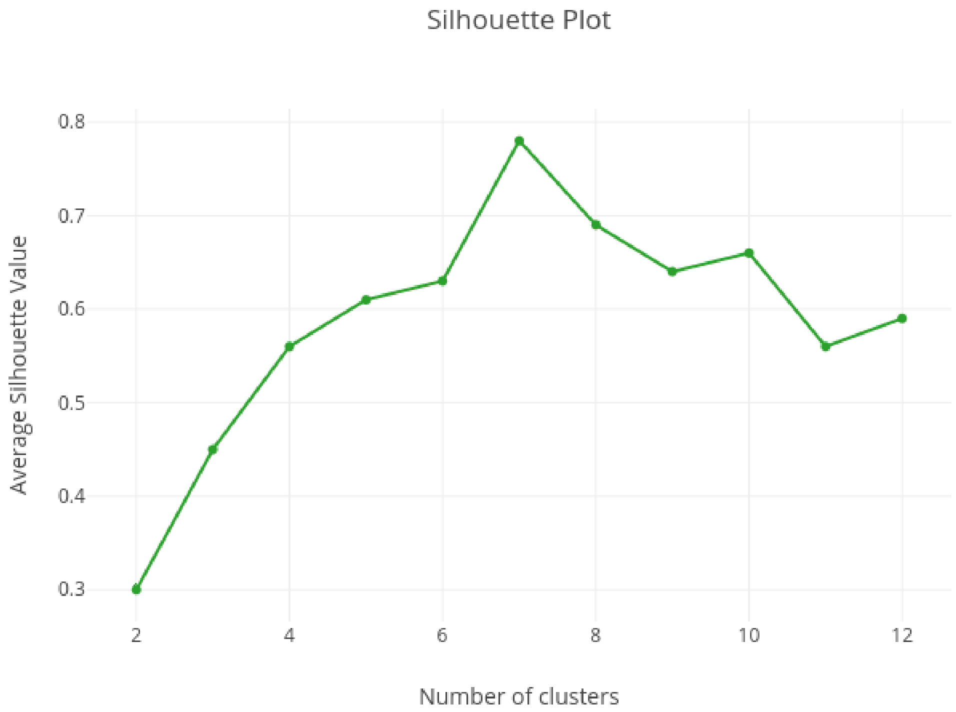


Figure 3.6. Silhouette Method

behavior. That is all the hosts in a cluster exhibit similar behavior. Recall the overarching premise of our work is to analyze the behavior of hosts looking at the aggregated host data. In order to analyze the behavior of hosts over a time period we should be able to compare hosts behavior across days. We dealt with this by initially comparing the hosts behavior for any two days and then extended it to any given time period. To achieve this we chose a reference day and the labeled the clusters on that day into an human understandable format.

In our case each cluster is exhibiting a unique behavior and hence comparing the hosts behavior turns out to be a problem of comparing clusters formed on two different days. Cluster comparison is not a well studied area. And, hence we translated this problem into an Assignment problem [17]. Assignment problem is a special type of linear programming problem which deals with the allocation of the various resources to the various activities on one to one basis. It does it in such a way that the cost or time involved in the process is minimum and profit or sale is maximum.

In our case both the resources and activities are nothing but clusters formed on two different days. Assignment Problem is a well studied problem and has polynomial time solutions. One such algorithm is the Hungarian algorithm.

3.3 Applications

Our system can be used to build different applciations that help in Network Management. Few of them are as follows: *Recommendation Systems*, We can capture the different behaviors exhibited by network users over time and use this to predict a behavior on a certain day of a week or month using the historical data. This helps network admin to prepare well ahead and plan the bandwidth or other network resources accordingly. *Dynamic FireWall Rules Generator*, Looking at the historic user behavior we can generate dynamic rules for security of the network. Few of them could be, to block certain ips that fall under Anomaly cluster or looking at the cluster centers and distribution of anomaly cluster we can change the authentication rules on number of unsuccessful attempts using different ports and so on.. *Anomaly detectors*, We can observe the change in cluster sizes and centers and there by notify the network admin if there is shift in behavior that network admin has to be notified.

CHAPTER 4

IMPLEMENTATION

In this chapter we will discuss about the implementation details of our system. Let us look at tools/techniques used at each step.

NetFlow Collection, The raw data collected at the routers is exported to NetFlow collectors as described in chapter 3. NFDUMP [] tools are used to process the netflow data. They support netflow version v5,v7 and v9. The two tools of primary importance are nfcapd(netflow capture daemon) that reads the netflow data from the network and stores the data into files. nfdump (netflow dump) reads the netflow data from the files stored by nfcapd. Using nfdump we import the data on to our local machines from the NetFlow Collectors. This imported data forms the primary source for our experiments. But this data is not in a usable format for applying data mining algorithms and hence we convert it into text format and save it in a csv file.

Feature Engineering, We used different python libraries such as NumPy, Spacy and Pandas to clean and manipulate data. Specifically, the mean and median calculating methods from Pandas for missing data and min max scaler respectively, log transformer for transformation from NumPy. For aggregating all the feature values by a host we used MongoDB contrary to keeping counters in the code. This has reduced the time taken for aggregation to under 2 minutes from 20 minutes for a days worth data. In this context we also wanted to try the approach of streaming algorithms such as bloom filters or count min sketch which gives the aggregate information with little memory foot print which we have explored a little but left for future work at this point.

Pattern Detection, As mentioned we have chosen K-Means for our pattern detector and to determine K we used elbow and silhouette approaches. Here we used scikit-learn package implementations of K-Means. To label the behaviors on a given day, we have selected a reference day and manually labeled its clusters mapping each cluster to a particular

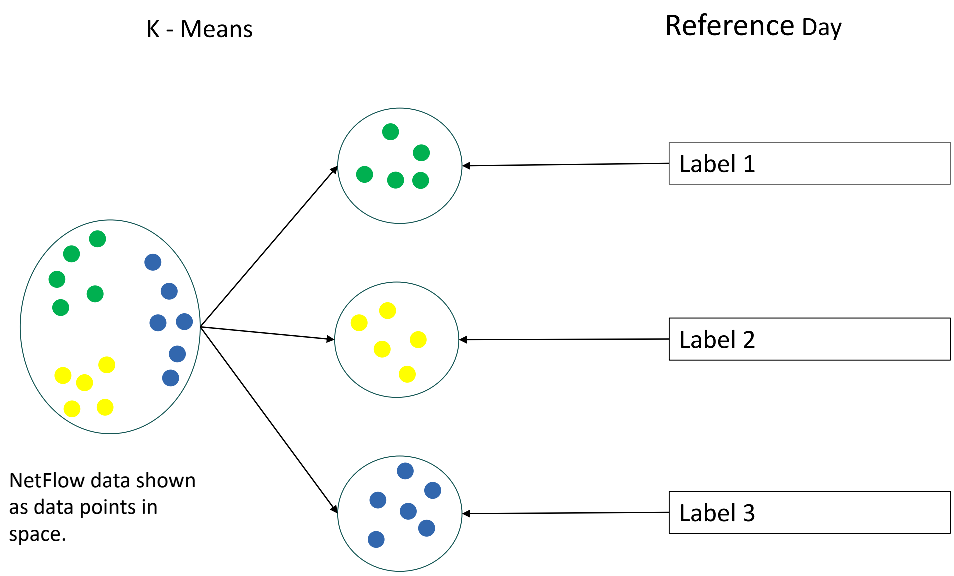


Figure 4.1. Process diagram on a reference day, with netflow eight dimensional data aggregated based on host shown as a 2D data point which when passed through our pattern detector split into three clusters. These clusters are manually inspected and labeled accordingly.

behavior as shown in **Figure 4.1** on the current page.

Now for every other day we compare the clusters formed on that given day with the reference day. Cluster comparison is an active research area and there are very few techniques to do this and hence we solved it by converting our cluster comparison problem in to an assignment problem. An assignment problem generally deals with assigning machines to tasks, workers to jobs etc.. The goal is to determine the optimum assignment that, for example, minimizes the total cost. The assignment problem is a fundamental problem in the area of combinatorial optimization. It has many polynomial time implementations and one of them is Hungarian Algorithm []. So, in our case the assignment problem is assigning the clusters on a given day to the clusters on the reference day. We use hungarian algorithm to solve this cluster assignment. As per this algorithm a cluster on a given day gets mapped to the cluster on a reference day based on the amount of effort it takes to move all the data points from this cluster to any of the reference day clusters center. Overall the goal is to make the assignments in such a way that the total effort is least. The least effort

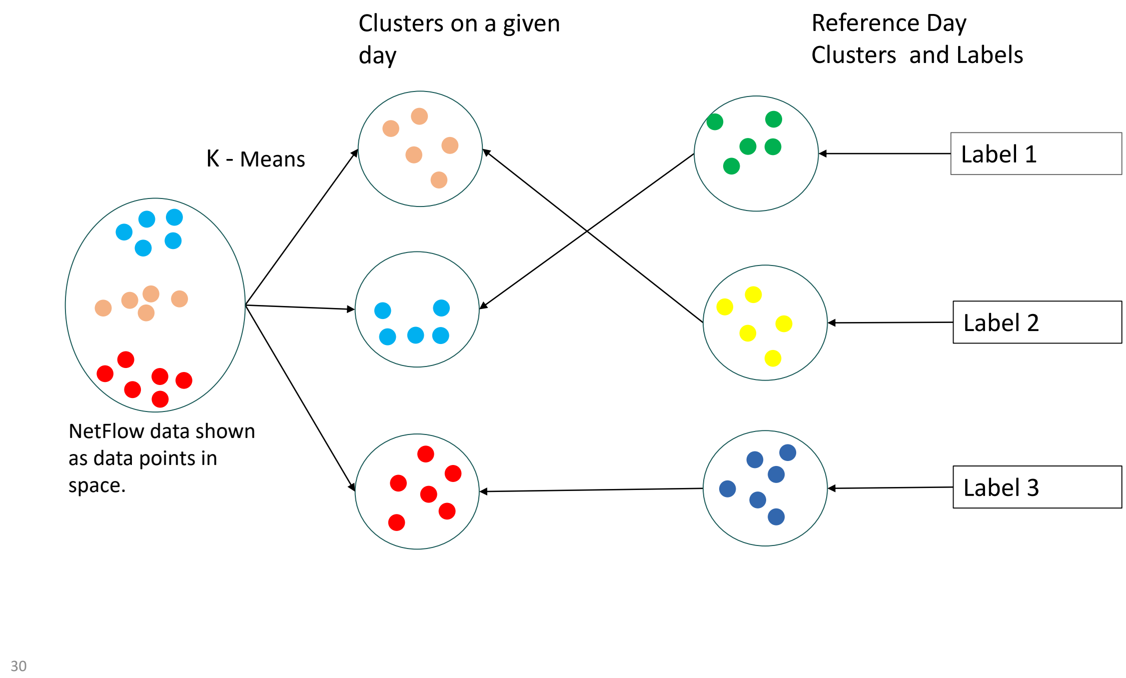


Figure 4.2. Process diagram on a reference day, with netflow eight dimensional data aggregated based on host shown as a 2D data point which when passed through our pattern detector split into three clusters. These clusters are manually inspected and labeled accordingly.

assignment for dec 12th is shown in the **Figure 4.2** on this page.

A point to be noted here is that we might need to change the reference day with time. This could be because of change in the usage of applications over time. Our recommendation is to choose a reference day that suggests the changed behavior and keep visiting this periodically.

CHAPTER 5

RELATED WORK

In this chapter, we review works that relate to usage of Machine Learning (ML)/Data Mining (DM) techniques in the area of networking and discuss in detail how researches are using these methods for different purposes. We also put in to the context how our problem statement and solution differ from the existing work and advances the state of the art.

First, we review the different ML/DM approaches that are being used for traffic classification and intrusion detection in the area of cyber security.

Second, we survey the vast body of work that does packet based inspection and also analyze the shift of research focus towards inspecting at higher granularities specifically the flow based inspection techniques.

Finally, we discuss how unsupervised approaches are being used to look at network data at higher granularities and compare our solution with the prior work.

5.1 Data Mining and Machine Learning Techniques for Intrusion Detection

There are many papers published in the area of cyber security describing the different ML/DM techniques used. Buczak et al [6] provided a survey of the popular techniques used and thoroughly describe ML/DM methods which provides a good starting point to people who intend to do research in the area of ML/DM for intrusion detection.

The survey by Buczak et al [6] has been mainly on explaining the ML and DM methods where as Bhuyan et al. [4] in his paper described different techniques used for network anomaly detection in detail. Garcia's work [14] also explained different intrusion techniques in detail. He extended his work to use machine learning techniques for anomaly detection with focus on signature based intrusion-detection. Narayan et al. [22] proposed a hybrid classification method utilizing both naive bayes and decision trees for intrusion detection. While their findings had higher accuracy applying these supervised techniques to our dataset was not feasible as we had very less labeled data.

Wired networks generally have multiple layers of security before the intruder enters the network. But, the wireless networks are more vulnerable to malicious attacks. They add a whole new semantics to the network security such as dynamically changing topology, different authentication techniques and ad-hoc formation of networks. The process followed in this paper works in the context of both wired and wireless networks. Zhang et al. in his work [27] provides a perspective focused only on wireless network protection.

5.2 Machine Learning for Traffic Classification

Apart from using ML/DM techniques for intrusion detection one other field where these have been extensively used is to classify the network data. The survey [21] lists the prominent ML/DM techniques used to classify the data and describes the need for traffic classification. In order to provide the promised quality of service and be liable to the government laws network administrators should be able to distinguish the traffic on their network.

Traditional well-known port number based classification is not sufficient on it's own to distinguish different applications as different services are using http protocol to send their data and obfuscate from the traffic classifiers. Also, papers such as choicenet [26] point out that to develop an economy plane for the internet similar to the implicit content based billing architecture in mobile architecture there is need for network administrators to classify the network data.

Naive Bayes form is the simplest technique used for this type of classification and has been explained in greater detail in the work of Andrew and Denis [19]. This work was extended by applying bayesian neural network approach [2] for increasing the accuracy. Though these classification techniques proved to be efficient in differentiating network data they have a drawback that they can only distinguish the network data into known classes which is in contrast to our goal of identifying implicit behavior which is not known in advance.

Renata[3] looked at unsupervised techniques for traffic classification. In contrast to the existing approaches he followed a principle of early detection. Accordingly, he looked at the first few packets of tcp flow and classified them into different applications. The underlying logic behind this method is that during handshake process each application behaves in a specific way exchanging a particular sequence of messages. He used K-Means algorithm to form clusters. Initially, training data collected over a period is used to generate a model which gives a set of clusters. When new data arrives simple Euclidean distance is used to map this flow to a cluster. The flow belongs to the cluster which it is closest to. Though, this approach had 80 accuracy it has few drawbacks, If we cannot capture initial few packets of a service it's effectiveness is compromised. If a flow doesn't fall under any cluster the behavior is undefined. Renata's [3] approach was similar to us as we also explored unsupervised techniques in our system but they differ in the point that they examine the packet data and map their clusters to only known classes of traffic.

Jeffrey and Mahanti et al. [11] explored hybrid techniques for traffic classification. The intuition behind this exploration is that not all data available will be labeled and when data from new services get appended to the existing dataset the supervised learning techniques are falling short in recognizing them and they map the new data set to one of the existing classes. To overcome this shortcoming they approached the problem in two steps. In the first step the dataset containing labeled and unlabeled data is passed to a clustering algorithm. In the second step the labeled data is used to classify clusters and this is done as follows: Within each cluster all the labeled data is considered and the label with majority forms the label of this cluster. Clusters without any labeled data are classified as 'Unknown'. When new data arrives it will be assigned based on the Euclidean distance to the closest cluster similar to [3] . This has combined advantages of supervised and unsupervised learning. It also decreases the training time because of few labeled data. It's uniqueness comes from being able to map the data from new applications and services to Unknown cluster or to their respective clusters if they are simple variation of existing application's characteristics. A slight variation of this approach has been used by us during our experiments and the following issues have been noticed. First, there were cases in which we have seen a cluster mapping to multiple labels as both turned out to be major labels in the cluster differing by a small value. In this case labeling a cluster just based on majority doesn't suggest the clusters behavior. Second, the amount of labeled data could be negligible compared to the size of the cluster. Third, two clusters could turn out to have similar labels. Lastly, the time consumed for this two step approach is high as we had to iterate through the data twice. Noticing that this technique isn't inline with our goals we have embraced a totally unsupervised approach for our problem.

5.3 Usage of flow records for IDS/Classification

Traditionally network data inspection is done by inspecting the contents of every packet. But, the granularity at which this is happening is changing based on the requirements. Earlier packet inspection is used to be a norm but inspecting the contents of every packet is prohibitive in this data-centric age. In this section we look at how flows (aggregated packet data) are being used as input for ML/DM algorithms in place of packets.

In our opinion, flow-based detection should not be seen as a replacement but should be treated as a complement to packet based inspection. We envision that a network administrator should be able to understand the behavior of his network through an aggregated view of network data (in our case flows) and should dive into packet data only when suspicious about a activity. This two step approach will ease the way the network administrators manage their network.

Work of Li et al. [12] and Gao et al. [13] are two examples of DOS detection using flows. In their approach a process tracks the presence of a flow in a specified time frame and an other process runs an anomaly-based engine that triggers if there is a sharp variation from the expected mean. A similar approach was proposed by Zhao et al.[28] to identify the IP addresses of the scanning hosts and DOS attackers using the flow data. Our system though uses flows captures a broader view apart from identifying these specific attacks. Network Flows are also used in building worm detectors [10] [9], botnet detectors [23] [18] [16]. Specifically, the botnet detector built by Livadas et al. [18] is of interest as they build a model using the aggregated flows similar to our work.

CHAPTER 6

RESULTS

In this chapter we evaluate our system using both labeled and unlabeled data. Our experiments in this section are conducted on a ubuntu16 virtual machine with 80 GB memory and 20 cpus of computing power.

6.1 Dataset: description and clustering

The traffic analyzed here is collected at the University of Utah's emulab routers [25]. This is a private repository that contains traces of bidirectional traffic into the emulab network and no payload. Traffic is collected on full duplex links at a speed of 1sGbps.

We passed the NetFlow records collected over the last six months of 2017 through our system to determine the set of behaviors hosts exhibited over this time. We have hand picked 7 days of this data for analysis. The chosen days are October 15, October 16, November 14, November 23, December 5, December 12, December 16[1]. We made sure that there is a mix of both week days , weekends, normal and heavy traffic days. Each days traffic is treated independently as the same host that appeared on two different days could exhibit different behaviors. For different evaluations in this section we used the data corresponding to the these days.

As mentioned earlier in the section 3.2.0.3 we have choosen a reference day for labeling the clusters. We applied following techniques to label the clusters on the reference day. The first one is port-based analysis to identify applications such as web, mail, and DNS. second one is the anomaly detection methods. Finally, a set of heuristic rules to label manually the behavior of the hosts. After the above steps we found the following clusters. Clusters labeled as T consists of hosts which use few ports on both sides to exchange interactive traffic. Clusters labeled as S consists of hosts which serve clients requests and conversely

---

[1]Days are picked without any prior knowledge of any metadata related to NetFlow records

clusters labeled C consists of hosts which request servers for information. Let us list the differences in these behaviors in detail.

Clusters T1-T2 are mostly one-to-one connections, the dominant traffic in these clusters is http/https, ssh and peer to peer traffic. P2P traffic is defined as traffic where hosts uses both TCP and UDP ports concurrently for communication, They also choose arbitary ports for communication. Further analysis, reveals that clusters are split based on packet size and flow size. While T1 has long living flows with large average packet sizes. T2 has large number of flows which are short lived and an average packet size less than T1.

Clusters C1-C2 contains hosts which behave as clients making connections with different servers, The cluster C2 is dominated by DNS traffic which is in accordance with the heavy DNS requests that the hosts residing in the emulab network make with the external servers (machines outside the Emulab Firewall). Cluster C1 on the other hand comprises of hosts that request for web pages and that have outbound mail traffic.

Clusters S1-S2 as mentioned above comprises of hosts that behave as servers which respond to client requests. These hosts generally communicate with different ports of destination machines from a fixed port. S1 cluster contains of majority of hosts which are within emulab network and are acting as servers accepting SSH requests. S2 cluster contains hosts that are both within and outside emulab network which are acting as servers for different sets of traffic such as SSH, Web and DNS which is further explained in the section 6.2.

Cluster A1 is the cluster that is of interest for network security as it contains of hosts that exhibit anamolous behavior trying to scan different access points to enter into the system.

The above description clearly points out that host based behavior extraction produces a richer and finer classification of host behaviors than a port based classifier. For example, Http traffic is split across different clusters T or S which represents how the same protocol can be used by hosts exhibiting different behaviors. Same is the case with SSH traffic. While most of these clusters contain hosts using different protocols. They are similar in the sense that each cluster exhibits different functional behavior such as Server, Client or One-to-One traffic.

We further examined the clusters to determine if there is any similarity in the way the

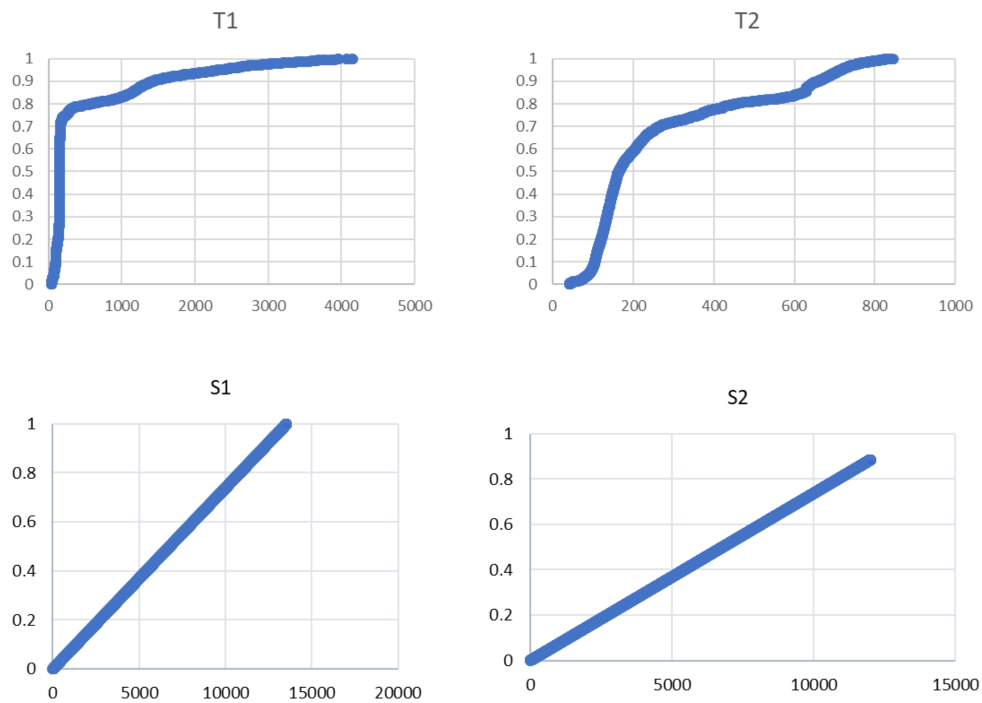
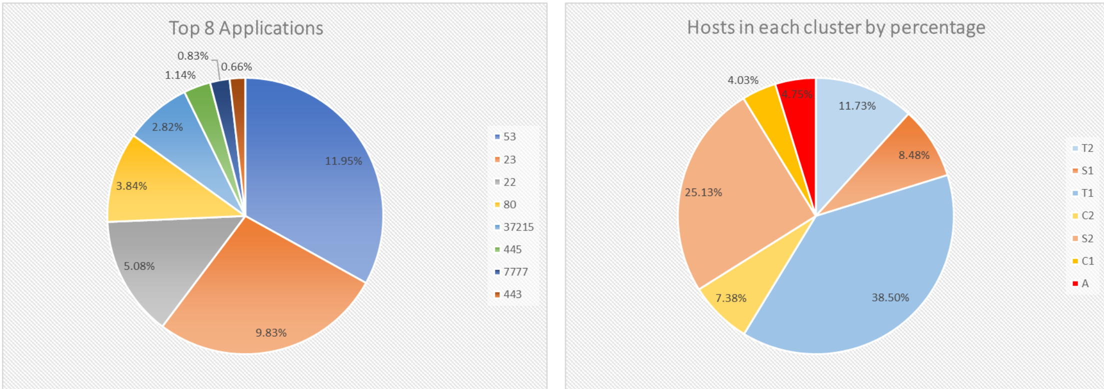


Figure 6.1. Average Bytes feature CDF for clusters T1,T2,S1,S2

clusters T1/T2 , C1/C2 and S1/S2 are formed. While we found that the clusters T1/T2 are distinguished by average byte size with hosts in T1 having an average byte size higher than hosts in the cluster T1, there were no such features that uniquely defined C1/C2 , S1/S2. This verifies the our system is not biased by any single feature.

The **Figure 6.2** on the next page is a comparison between the classification made by a traditional port based classifer 6.1(a) and our host based classifier 6.1(b). The traditional port based classifier identified the top 8 applications used by the hosts in the system. DNS(53) is the most used application with 11.95% of the whole traffic trying to query DNS servers. It is followed by Telnet(23), SSH(22) with a share of 9.83% and 5.08% traffic respectively. We can also see web traffic both secured and unsecured in the top 8 accounting to 3.84%, 0.66% traffic. While, we can see the applications that accounted to majority of traffic by traditional classifier, Our system on the other hand provided an alternate view for the same NetFlow data by extracting the behaviors these hosts are exhibting. From the **Figure 6.2** on the following page we can see that, hosts that are behaving as clients



6.1 (a)   6.1 (b)

Figure 6.2. Comparison of hosts using port based classifier (left) and host behavior extractor (right) on December 12, 2017.

account to 11.41% (both C1 and C2 combined) of traffic and hosts that are behaving as servers account to 33.61% (both S1 and S2 combined). The majority of the traffic(50.23%) comprises of interactive communication between hosts while the anomalous hosts trying to invade into the network form the minority(4.75%).

6.2 Cross Validation

We compared the results obtained from our system with other classification techniques to better understand the significance of our approach.

Cross Validation with a port based classifier, We used a classic port based classifying technique to cross validate our results. Though, it isn't a perfect measure and fails in many cases it is good enough to classify a host to a particular class in most of the cases. In the scenarios when this technique cannot classify a host to particular class we label it as a 'Mix' traffic. The heuristic that we used for labeling as 'Mix' traffic is when the dominant class accounts for less than 50 percent of the traffic by that host. Applying this procedure we observed 20 different classes of traffic, out of which the most frequently observed are discussed here: HTTP, DNS, SSH, MAIL, TELNET, FTP, CHAT, SCAN, SMTP, MIX. Most of these classes are self explanatory based on the ports they operate. SCAN is a class that is different from others while all the other classes deal with single port SCAN deals with multiple ports. A host falls under SCAN category if majority of its traffic is trying to hit multiple ports on a system trying to find an open port to infiltrate. The cross-validation between the port based approach and our procedure on one of the choosen day December 12th is reported in **Table 6.1** on the current page.

The row headers in the table correspond to the lables of the clusters generated by our system while the column headers correspond to the different classes of traffic derived by a port based classifier as mentioned above. Each row in the table describes the percentage of hosts within each cluster that fall under different classes of traffic as classified by a port based classifier. For example, in the cluster T1 we have 60.88% of hosts that are using web applications, 0.86% of hosts are involved in DNS traffic, 15.03% of hosts exchange SSH, 0.72% exchange TELNET traffic and 22.04% of the hosts in this cluster are classified as MIX traffic. Some important observations from this table are:

- From the bolded parts in the table we can safely conclude that we have an high match of host classification which reflects the competence of the proposed procedure.
- It is also clearly evident from here that the nature of clusters described earlier confirm

Table 6.1. Cross-valdation of the host behavior extraction with port based analysis.

Label	WEB	DNS	SSH	TELNET	SMTP	FTP	SCAN	MIX	# Hosts
T1	**60.88%**	0.86%	15.03%	0.72%	0.36%	0.02%	0	22.04%	14526
T2	8.29%	0.67%	27.27%	1.01%	0.60%	0	0	**62.10%**	3375
C1	**40.12%**	0.92%	0	0	**41.31%**	5.13%	0	12.52%	1159
C2	2.14%	**90.82%**	3.05%	0	2.74%	0.07%	0.10%	1.18%	2123
S1	1.64%	8.37%	**78.50%**	10.45%	1.03%	0	0	3.51%	2439
S2	15.45%	38.98%	1.40%	0	16.52%	0	0	27.65%	4927
A1	2.78%	12.12%	2.31%	6.98%	0.73%	0.91%	**64.91%**	9.26%	217

with the results obtained with cross-validation.

- This also suggests that all the hosts that fall into a particular class based on port based analysis are not necessarily in the same cluster. They are dispersed across the clusters which shows the necessity of a host based behavior extraction.

For example, cluster S1 clearly confirms the fact that the hosts in this cluster are dealing with SSH traffic and the same traffic is also present in T1 and T2 clusters. Similary, hosts in the cluster S2 contains of servers that provide responses for web, ssh and other requests. This cluster has similar distribution as cluster C1 but they act exactly opposite like servers and clients respectively. Also, few points of observation from the table are, T1 has mostly requests and responses between hosts spread across different applications. hosts which are grouped as T2 have most of it's hosts labeled as Mix, which on further investigation revealed that many of the hosts in this class don't pass the majority test to be distinguished into any particular class. Finally, Scanners distribution was minimal across all the clusters. The sparsity of this table also serves as a confident measure for our proposed approach.

6.3 Labeled Data: description and clustering

What is this section about? In this section we are evaluating our system to determine its accuracy and precision on identifying host behaviors.

How is it different from previous evaluations? While the previous evaluations in this chapter used unlabeled data to validate our system. Here, we are using labeled data to do this evaluation.

So, where did you get the labeled data? The labeled data that we used in this section is provided by Center for Applied Internet Data Analysis (CAIDA).

What is CAIDA? CAIDA is a research group based at the San Diego Supercomputer Center (SDSC) on the UC San Diego campus who conduct network research and build research infrastructure to support large-scale data collection, curation, and data distribution to the scientific research community.

How did CAIDA get its labeled data and explain the features of this data set? CAIDAs dataset consists of thirteen different scenarios. Each scenario is a capture of network data written into a pcap file. Each capture consists of both real network traffic and the synthetic traffic they have generated to emulate that scenario. For our evaluation purpose we are considering four of these scenarios.

- In the first scenario they have generated traffic to emulate the situation of scanners in the network and hence their capture consists of two classes of traffic namely, Normal and Scanning web proxies.
- In the second scenario they have generated traffic to emulate the situation of chinese attackers in the network and hence their capture consists of two classes of traffic namely, Normal and Chineses hosts.
- The third scenario consists of synthesized traffic to emulate hosts exhibiting multiple behaviors in the network namely, Normal, Background and UDP& ICMP DDOS attacks.
- Fourth scenario consists of synthesized traffic to emulate hosts exhibiting multiple behaviors in the network namely, Normal, Background and UDP& ICMP DDOS attacks.

These captured pcap files are further processed to obtain information about NetFlows. **Table 6.2** on the following page represents the duration of each scenario and the contents of the pcap file such as the number of packets, the number of NetFlows and the size of the file itself. The distinctive character of this data set is that the data captured in each scenario is manually examined and labeled. The labeled dataset looks similar to the one which we capture at the emulab router excpet for the last column where the flow is labeled which can be seen in the **Figure 6.3** on the next page.

Using the different captures in **Table 6.2** on the following page we evaluated our systems basic functionality such as how effectively our system is able to extract the host behaviors, the detection rate of the different possible attacks and how will it respond with scale (when there are multiple users on multiple networks communicating simultaneously). The results are as follows:

6.3.1 Scenario 1:

The scenario 1 is of duration 11.63 hours with 37.6 GB of data. This dataset contains hosts exhibiting two unique behaviors namely normal traffic and scanners as labeled by

Table 6.2. Amount of data on each scenario.

Id	Duration(hrs)	# Packets	#NetFlows	Size
1	11.63	4,481,167	129,833	37.6 GB
2	0.38	7,467,139	114,078	5.8 GB
3	4.21	62,089,135	1,121,077	53 GB
4	66.85	167,730,395	4,710,639	121 GB

the manual inspection. Through this experiment we would like to check our systems basic functionality of identifying different behaviors. When we passed this data through our sytsem we found formation of two major clusters and the results are as shown in **Table 6.3**

```
StartTime,Dur,Proto,SrcAddr,Sport,Dir,DstAddr,Dport,State,sTos,dTos,TotPkts,TotBytes,SrcBytes,Label
2011/08/15 10:42:52.613616,1065.731934,udp,188.114.23.248,14268,  <->,147.32.84.118,1150,CON,0,0,2,252,145,flow=Background-UDP-Established
2011/08/15 10:42:52.676517,1471.787109,udp,2.224.28.134,48279,  <->,147.32.84.118,1150,CON,0,0,2,252,145,flow=Background-UDP-Established
2011/08/15 10:43:34.544316,1095.947266,udp,41.145.83.148,40812,  <->,147.32.84.118,1150,CON,0,0,2,288,145,flow=Background-UDP-Established
2011/08/15 10:58:26.370803,2896.377197,udp,217.42.33.104,55066,  <->,147.32.84.118,1150,CON,0,0,3,433,290,flow=Background-UDP-Established
2011/08/15 10:58:27.208147,1384.708008,udp,41.107.0.220,11066,  <->,147.32.84.118,1150,CON,0,0,2,288,145,flow=Background-UDP-Established
2011/08/15 10:59:02.040572,1071.322388,udp,213.44.235.37,9200,  <->,147.32.84.118,1150,CON,0,0,2,288,145,flow=Background-UDP-Established
2011/08/15 11:00:05.704408,3582.270264,tcp,147.32.84.134,45266,  <?>,205.188.10.203,443,PA_PA,0,0,662,70674,25955,flow=From-Normal-V45-Jist
2011/08/15 11:00:05.704760,44.728664,tcp,62.44.1.18,80,  <?>,147.32.84.118,51463,PA_FRA,0,0,768,756975,741555,flow=Background
2011/08/15 11:00:05.707670,0.000000,udp,147.32.80.9,53,   ->,147.32.85.103,18006,INT,0,,1,175,175,flow=From-Normal-V45-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.709591,0.003098,udp,147.32.85.103,41094,  <->,147.32.80.9,53,CON,0,0,2,223,71,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.715153,0.080609,udp,147.32.85.103,27423,  <->,147.32.80.9,53,CON,0,0,2,212,73,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.716016,35.265488,tcp,212.194.183.145,49584,  <?>,147.32.85.56,44076,FPA_FPA,0,0,43,3147,2061,flow=Background
2011/08/15 11:00:05.718808,136.336746,tcp,74.63.47.82,80,  <?>,147.32.84.59,60106,PA_RA,0,0,2451,2319965,2272325,flow=Background-Established-cmp
2011/08/15 11:00:05.721037,0.000273,udp,147.32.86.20,62510,  <->,147.32.80.9,53,CON,0,0,2,284,76,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.721591,0.888468,tcp,147.32.86.20,1962,   ->,193.93.174.21,80,FSPA_FSPA,0,0,99,92657,1946,flow=Background-TCP-Established
2011/08/15 11:00:05.722349,3599.194336,tcp,109.80.225.83,49708,  <?>,147.32.84.171,57116,PA_A,0,0,79509,59759658,57384858,flow=Background
2011/08/15 11:00:05.724203,0.000256,udp,147.32.84.222,55073,  <->,147.32.80.9,53,CON,0,0,2,210,80,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.724587,0.000241,udp,147.32.84.222,39714,  <->,147.32.80.9,53,CON,0,0,2,236,93,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.724940,0.000302,udp,147.32.84.222,51315,  <->,147.32.80.9,53,CON,0,0,2,282,80,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.727892,0.000000,udp,129.6.15.28,123,   ->,147.32.87.135,4927,INT,0,,1,90,90,flow=Background-UDP-Attempt
2011/08/15 11:00:05.728471,0.121183,tcp,147.32.84.59,36828,  <?>,75.101.227.132,80,FA_A,0,0,2,120,60,flow=Background-Established-cmpgw-CVUT
2011/08/15 11:00:05.729558,0.000313,udp,147.32.84.36,43561,  <->,147.32.80.9,53,CON,0,0,2,313,85,flow=To-Background-UDP-CVUT-DNS-Server
2011/08/15 11:00:05.730093,0.000264,udp,147.32.84.36,39498,  <->,147.32.80.9,53,CON,0,0,2,288,83,flow=To-Background-UDP-CVUT-DNS-Server
```

Figure 6.3. Manually labeled synthetic data.

from this matrix:

- There are two possible predicted behaviors: 'Scanning Web Flows' and 'Normal Flows'.
- Our system made a total of 129,833 predictions.
- Out of those our system predicted 46,490 flows are from hosts which are scanning and 83,343 flows are from hosts which are exhibiting Normal behavior.
- In reality 84,239 flows are from hosts that show Normal behavior and 45,594 flows are from hosts that are scanners.

Now, let us calculate some numbers that describe the performance of our system.

- **Accuracy:** Overall, how often is our system identifying correctly?
  - Number of flows it as predicted as scanners and they actually are flows belonging to scanners summed up with Number of flows it as predicted as Normal

Table 6.3. Scenario 1.

N= 129833		**Predicted**	
		Scanning Web Flows	Normal Flows
Actual	Scanning Web Flows	42,503	3,091
	Normal Flows	3,987	80,252
		46,490	83343

and they actually are flows belonging to Normal hosts divided by the total predictions made by our system.

  - (42503+80252)/129833 = 0.9454

- **Misclassification Rate:** Overall, how often is our system wrong?

  - Sum of number of flows our system has predicted wrong across all the behaviors divided by total predictions made by our system. In this case, number of flows it has predicted as scanners when they are actually flows belonging to Normal hosts summed up with number of flows it has predicted as Normal when they actually belong to Scanning hosts divided by the total predictions made by our system.

  - (3987+3091)/129833 = 0.0545

- **Precision:** For a certain behavior, how often it is identifying correctly?

  - The number of flows for a certain behavior that are identified correctly divided by total number of flows predicted for that behavior. Number of flows which are correctly predicted as scanners divided by total number of flows predicted as scanners, Number of flows which are correctly predicted as Normal divided by total number of flows predicted as Normal.

  - (42503)/46490 = 91.4% precise in finding scanners.

  - (80252)/83343 = 96.2% precise in finding normal hosts.

From this we learn that our system can identify unique behaviors with high accuracy and precision.

6.3.2 Scenario 2:

The scenario 2 is of duration 0.38 hours with 5.8 GB of data. This dataset contains hosts exhibiting two unique behaviors namely normal traffic and chinese hosts as labeled by the manual inspection. This experiment is aimed to determine the basic functionality of our system as scenario 1 and to determine the rate of detection of chinese hosts which are prevalent in network attacks in our real data set collected at Emulab routers. When we passed this data through our system we found formation of two major clusters and the results are as shown in **Table 6.4** on this page. This information that we can learn from the confusion matrix is:

- There are two possible predicted behaviors: 'Chinese Host Flows' and 'Normal Flows'.
- Our system made a total of 107,978 predictions.
- Out of those our system predicted 28,368 flows are from hosts which are hosted in china and 79,610 flows are from hosts which are exhibiting Normal behavior.
- In reality 82,625 flows are from hosts that show Normal behavior and 25,353 flows are from hosts that are hosted in chinese.

For this scenario the performance numbers are as follows:

- **Accuracy:** Overall, how often is our system identifying correctly?
  - Number of flows it as predicted as from hosts hosted in chinese and they actually are flows belonging to chinese hosts summed up with Number of flows it

Table 6.4. Scenario 2.

N= 107,978		Predicted	
		Chinese Hosts Flows	Normal Flows
Actual	Chinese Hosts Flows	22,456	2,897
	Normal Flows	5,912	76,713
		28,368	79,610

as predicted as Normal and they actually are flows belonging to Normal hosts divided by the total predictions made by our system.

  - (22456+76713)/107978 = 0.89184

- **Misclassification Rate:** Overall, how often is our system wrong?

  - Number of flows it as predicted as chinese hosts and they are not flows belonging to chinese hosts along with Number of flows it as predicted as Normal and they are not flows belonging to Normal hosts divided by the total predictions made by our system.
  - (5912+2897)/107978 = 0.117

- **Precision:** For a certain behavior, how often it is identifying correctly?

  - Number of flows which are correctly predicted as chinese hosts divided by total number of flows predicted as chinese hosts, Number of flows which are correctly predicted as Normal divided by total number of flows predicted as Normal.
  - (22456)/28368 = 76.1% precise in finding chinese hosts.
  - (76713)/79610 = 96.3% precise in finding normal hosts.

6.3.3 Scenario 3:

The scenario 3 is of duration 4.21 hours with 53 GB of data. This dataset contains hosts exhibiting three unique behaviors namely normal ,background traffic and flows from hosts trying to do a DDOS as labeled by the manual inspection. When we passed this data through our system we found formation of three major clusters and the results are as shown in **Table 6.5** on the next page. This information that we can learn from the confusion matrix is:

- There are two possible predicted behaviors: 'Background Flows' ,'Normal Flows', 'UDP & ICMP DDOS Flows'.
- Our system made a total of 1,111,836 predictions.

- Out of those our system predicted 689,249 flows are from hosts which are exhibiting Normal behavior , 73,650 flows are from hosts which are trying to perform DDOS attack and 348,937 flows as Background flows.
- In reality the flows from normal hosts, background hosts and DDOS attackers are 711459, 332104 and 68273 respectively.

For this scenario the performance numbers are as follows:

- **Accuracy:** Overall, how often is our system identifying correctly?
  - (681167+66377+320011)/1111836 = 0.9601
- **Misclassification Rate:** Overall, how often is our system wrong?
- **Precision:** For a certain behavior, how often it is identifying correctly?

The following evaluations also validate our systems results. Our observation is that the behaviors exhibited by the hosts are quite stable over several months which can be seen

Table 6.5. Scenario 3.

N = 1,111,836		Predicted		
		Normal Flows	UDP & ICMP DDOS Flows	Background Flows
Actual	Normal Flows	681,167	3,123	27,169
	UDP & ICMP DDOS Flows	139	66,377	1,757
	Background Flows	7,943	4,150	320,011
		689,249	73,650	348,937

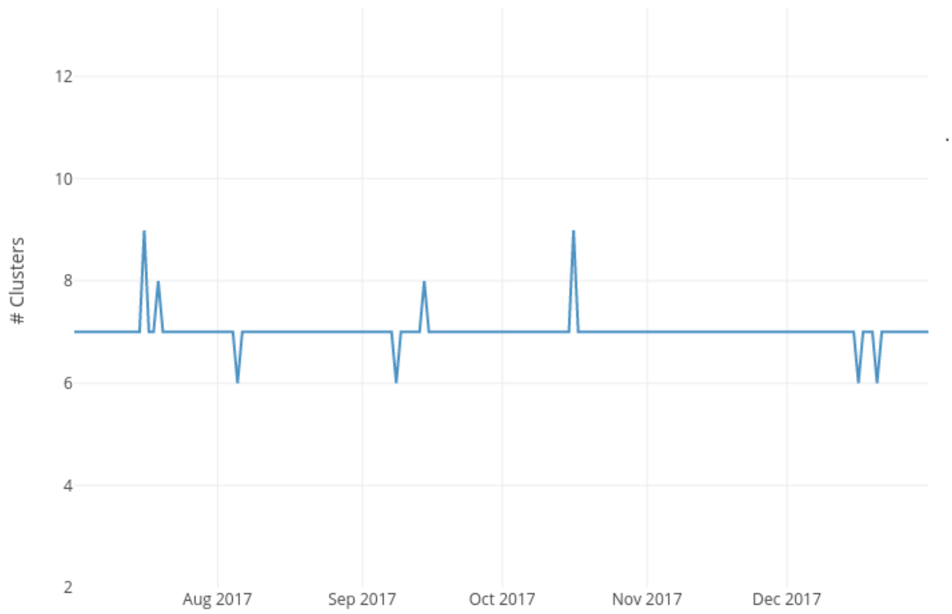


Figure 6.4. Plot of number of clusters formed over a time period.

in the **Figure 6.4** on this page. The graph is a representation of the behaviors exhibited by hosts over a six months of time period. Since we map each cluster to a behavior this is equals to the number of clusters that are formed over a time period. As, we are using K-means clustering, in our case this is a graph drawn between the value of K on X-axis and the day on which we found this value on Y-axis. Except for few points it is a straight line indicating the constants behaviors that hosts exhibited in our system. Our recommendation is to update the host behaviors every couple of months or as per the change in usage of applications.

REFERENCES

[1] D. Arthur and S. Vassilvitskii, *k-means++: The advantages of careful seeding*, in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[2] T. Auld, A. W. Moore, and S. F. Gull, *Bayesian neural networks for internet traffic classification*, IEEE Transactions on neural networks, 18 (2007), pp. 223–239.

[3] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, *Traffic classification on the fly*, ACM SIGCOMM Computer Communication Review, 36 (2006), pp. 23–26.

[4] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, *Network anomaly detection: methods, systems and tools*, Ieee communications surveys & tutorials, 16 (2014), pp. 303–336.

[5] C. Boutsidis, P. Drineas, and M. W. Mahoney, *Unsupervised feature selection for the k-means clustering problem*, in Advances in Neural Information Processing Systems, 2009, pp. 153–161.

[6] A. L. Buczak and E. Guven, *A survey of data mining and machine learning methods for cyber security intrusion detection*, IEEE Communications Surveys & Tutorials, 18 (2016), pp. 1153–1176.

[7] M. Dash, K. Choi, P. Scheuermann, and H. Liu, *Feature selection for clustering-a filter solution*, in Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, IEEE, 2002, pp. 115–122.

[8] K. G. Derpanis, *K-means clustering*, 2006.

[9] T. Diibendorfer and B. Plattner, *Host behaviour based early detection of worm outbreaks in internet backbones*, in Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005. 14th IEEE International Workshops on, IEEE, 2005, pp. 166–171.

[10] T. Dubendorfer, A. Wagner, and B. Plattner, *A framework for real-time worm attack detection and backbone monitoring*, in Critical Infrastructure Protection, First IEEE International Workshop on, IEEE, 2005, pp. 10–pp.

[11] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, *Semi-supervised network traffic classification*, in ACM SIGMETRICS Performance Evaluation Review, vol. 35, ACM, 2007, pp. 369–370.

[12] Y. Gao, Z. Li, and Y. Chen, *Towards a high-speed routerbased anomaly/intrusion detection system*, Technical Report, (2005).

[13] ——, *A dos resilient flow-level intrusion detection approach for high-speed networks*, in Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on, IEEE, 2006, pp. 39–39.

[14] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, *Anomaly-based network intrusion detection: Techniques, systems and challenges*, computers & security, 28 (2009), pp. 18–28.

[15] E. L. Johnson and H. Kargupta, *Collective, Hierarchical Clustering from Distributed, Heterogeneous Data*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 221–244.

[16] A. Karasaridis, B. Rexroad, D. A. Hoeflin, et al., *Wide-scale botnet detection and characterization.*, HotBots, 7 (2007), pp. 7–7.

[17] H. W. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistics (NRL), 2 (1955), pp. 83–97.

[18] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer, *Usilng machine learning technliques to identify botnet traffic*, in Local Computer Networks, Proceedings 2006 31st IEEE Conference on, IEEE, 2006, pp. 967–974.

[19] A. W. Moore and D. Zuev, *Internet traffic classification using bayesian analysis techniques*, in ACM SIGMETRICS Performance Evaluation Review, vol. 33, ACM, 2005, pp. 50–60.

[20] F. Murtagh, *A survey of recent advances in hierarchical clustering algorithms*, The Computer Journal, 26 (1983), pp. 354–359.

[21] T. T. Nguyen and G. Armitage, *A survey of techniques for internet traffic classification using machine learning*, IEEE Communications Surveys & Tutorials, 10 (2008), pp. 56–76.

[22] S. Peddabachigari, A. Abraham, C. Grosan, and J. Thomas, *Modeling intrusion detection system using hybrid intelligent systems*, Journal of network and computer applications, 30 (2007), pp. 114–132.

[23] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, *Botnet detection based on network behavior*, in Botnet detection, Springer, 2008, pp. 1–24.

[24] T. M. Thang and J. Kim, *The anomaly detection by using dbscan clustering with multiple parameters*, in Information Science and Applications (ICISA), 2011 International Conference on, IEEE, 2011, pp. 1–5.

[25] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, *An integrated experimental environment for distributed systems and networks*, in Proc. of the Fifth Symposium on Operating Systems Design and Implementation, Boston, MA, Dec. 2002, USENIX Association, pp. 255–270.

[26] T. Wolf, J. Griffioen, K. L. Calvert, R. Dutta, G. N. Rouskas, I. Baldin, and A. Nagurney, *Choicenet: toward an economy plane for the internet*, ACM SIGCOMM Computer Communication Review, 44 (2014), pp. 58–65.

[27] Y. Zhang, W. Lee, and Y.-A. Huang, *Intrusion detection techniques for mobile wireless networks*, Wireless Networks, 9 (2003), pp. 545–556.

[28] Q. Zhao, J. Xu, and A. Kumar, *Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation*, IEEE Journal on Selected Areas in Communications, 24 (2006), pp. 1840–1852.