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 2017

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Teja Kommineni**

has been approved by the following supervisory committee members:

Robert Ricci, Chair(s) __
Date Approved

Kobus van der Merwe, Member __
Date Approved

Suresh Venkatasubramanian, Member __
Date Approved

__, Member __
Date Approved

__, Member __
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

Understanding the behaviour of flows at port level is well studied whereas understanding the behaviour of hosts and groups of hosts is less well studied. The latter 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 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.

TIMELINE

This section gives an overview of the timeline associated with the work involved in the thesis.

Task 1	Extracting the Netflow data from University Network	Jan 2017	Done
Task 2	Preprocessing of Netflow data for Data Mining purposes.	Mar 2017	Done
Task 3	Evaluate different Data Mining approaches for extracting host behaviors	Apr 2017	Done
Task 4	Built a tool to analyze extracted host behaviors	Aug 2017	Done
Task 5	Researching state of the art algorithms for cluster comparison	Oct 2017	Done
Task 6	Write up Abstract	Oct 2017	Done
Task 7	Finish Writing Chapter on Introduction	Nov 2017	Done
Task 8	Changing the system design slightly to determine when network admins have to define new host behaviors	Nov 2017	In Progress
Task 9	Finish the chapter on Pattern Detector	Nov 2017	In Progress
Task 10	Defend the thesis proposal	Nov 2017	In Progress
Task 11	Prepare controlled setting with network data	Dec 2017	TBD
Task 12	Evaluate performance of the system and compare with existing tools.	Dec 2017	TBD
Task 13	Finish writing evaluation chapter	Jan 2018	TBD
Task 14	Finish writing conclusion	Jan 2018	TBD
Task 15	Final thesis defense	Feb 2018	TBD

Table 0.1. Timeline of tasks involved in thesis.

CONTENTS

CHAPTERS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Understanding the behaviour of flows at port level is well studied whereas understanding the behaviour of hosts and groups of hosts is less well studied. The latter 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 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. This behavior extraction helped us in gaining interesting insights into our network data. Also, as we can see the amount of data is increasing daily and hence storing and traversing through the data is difficult that is why we approached data mining technique which is a common approach these days to deal with huge amount of data.

Presently, admins use different tools to mange their networks which provide network-related metrics by default. It is interesting to know about amount of bytes being transferred on the wire but what is more interesting is knowing that half of your traffic is comprised of DNS requests or one fourth of the traffic contains scanners trying to gain access of systems within the network. This is extremely valuable information and serious network monitoring tools should provide this information in addition to the general metrics provided by them such as inbound and outbound traffic, dropped packets and others. Some examples of behaviors that system administrators would be interested in knowing 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.

Capacity Planning, Traffic Classification, Security Management are few of day to day activities performed by network administrator. We explain here how extracting host behaviors could help network admins in performing these network management activities in an efficent way. Network capacity planning is the process of ensuring that sufficient bandwidth is provisioned such that the committed core network service-level agreements can be met. Generally, core capacity planning is triggered by the admins when they observe that the utilization statistics of a network have crossed a threshold value or when they have data that suggest that congestion issues are likely to occur in the network. But there are a few consequences to this approach, firstly, following a rule based approach without understanding the total network demands will not suffice. There could be situations where even doubling the capacity of network when it is being used at half its capacity will not guarantee SLAs. Secondly, rule based provisioning could lead to over provisioning.Our approach of examining Host level behavior helps in understanding the network-wide traffic demands there by ensuring capacity planning. It helps the network admin to understand the types of host behaviors that are leading to congestion. A lot of bandwidth growth is among hosts that behave same way. And a network operator one might want to decide if the traffic is legitimate or not and act upon. A legitimate case would be when hosts are trying to backup to a server and the network is filled to its capacity. In this case the capacity of network has to be increased. There could be an other situation where there is high amount if bi-directional traffic and these hosts could be bit-torrent users in this case instead of increasing the capacity of network we would decrease it or take suitable action.

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 to 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 and Data Mining 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 implicit, previously unknown and potentially useful information from data. 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 the business or problem variable that experts assume 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.

2.0.1 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 [19], these models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. Examples of these models are hierarchical clustering algorithm and its variants.

Centroid models, These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. K-Means clustering algorithm [7] is a popular algorithm that falls into this category. In these models, the number of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optimum.

Distribution models[14], These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). A popular example of these models is Expectation-maximization algorithm which uses multivariate normal distributions.

Density Models [23], These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in 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 clustering [17] is a clustering analysis algorithm that groups objects based on their feature values into K disjoint clusters. Objects that are classified into the same cluster have similar feature values. K is a positive integer number specifying the number of clusters, and has to be given in advance. Here are the four steps of the K-means clustering algorithm:

- Define the number of clusters K.
- Initialize the K cluster centroids. This can be done by arbitrarily dividing all objects into K clusters, computing their centroids, and verifying that all centroids are different from each other. Alternatively, the centroids can be initialized to K arbitrarily chosen, different objects.
- Iterate over all objects and compute the distances to the centroids of all clusters. Assign each object to the cluster with the nearest centroid.
- Recalculate the centroids of both modified clusters.
- Repeat step 3 until the centroids 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 = (x1, ..., xm) and y = (y1, ..., ym) are two input vectors with m quantitative features. In the Euclidean distance function, all features contribute equally to the function value. However, since different features are usually measured with different metrics or at different scales, they must be normalized before applying the distance function.

CHAPTER 3

DESIGN OVERVIEW

Detecting the behaviors implicit in the netflow data and using them to build systems that 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 four essential components: Netflow Collection, Feature Engineering, Pattern Detector, and applications as shown in figure 3.1.

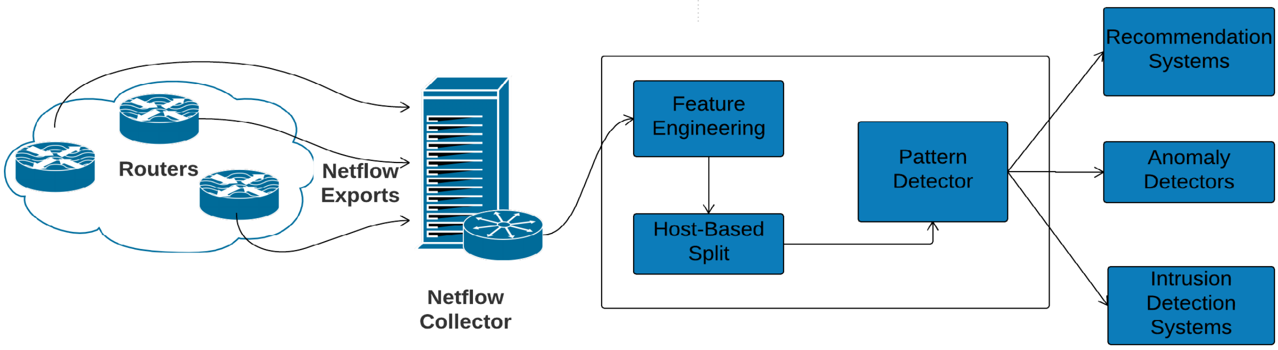


Figure 3.1. Architecture.

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 servers 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.

What is NetFlow

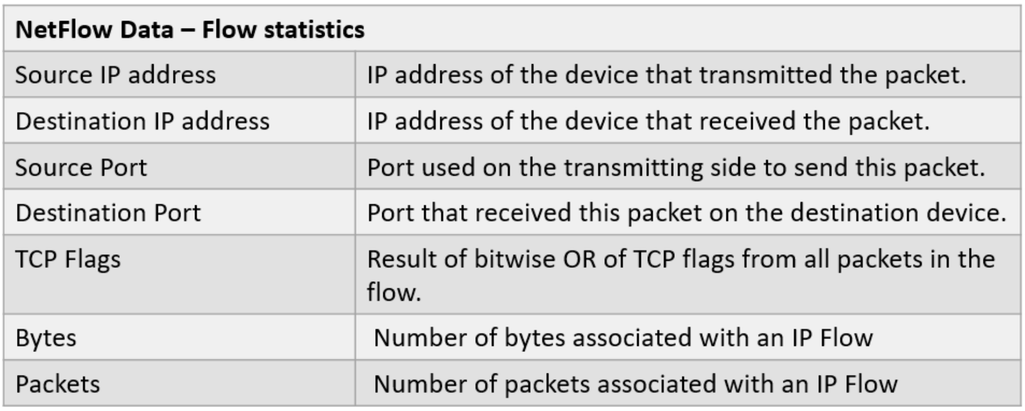
NetFlow is a proprietary protocol for collecting IP flow packets information on networks. Though some variant versions exist, a NetFlow record summarizes a network traffic flow as its source and destination IP addresses, source and destination ports, transportation protocol as well as the traffic volume transmitted during this flow session. NetFlow operates by creating new flow cache entries when a packet is received that belongs to a new flow. Each flow cache keeps track of the number of bytes and packets of similar traffic during certain period of time until the cache expires, then this information is exported to a collector.

The captured flows are exported to NetFlow collectors at frequent intervals. The flows that are to be exported are determined based on the following rules: 1) when it 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 ??** on page ??.

3.1.2 Feature Engineering

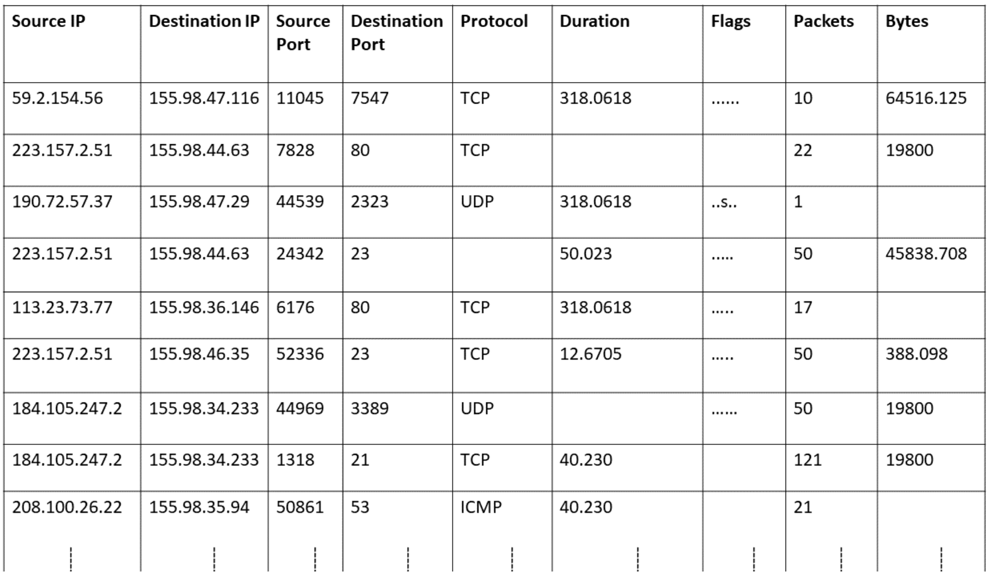
Feature Engineering is the process of transforming raw data into features that better represent the underlying structure to the models so that they increase the accuracy of the model when applied on unseen data. This step comes before modeling and after seeing the data. Features extracted from the data 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 ML/DM approaches are a factor of the data set in hand, the features generated , the prediction model and the way the problem is framed.



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

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.



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	

Table 3.1. Netflow Raw Data.

3.1.2.1 Missing Data

Table 3.1 on the current page is a sample of network data collected using netflow. In total there are 40 columns 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

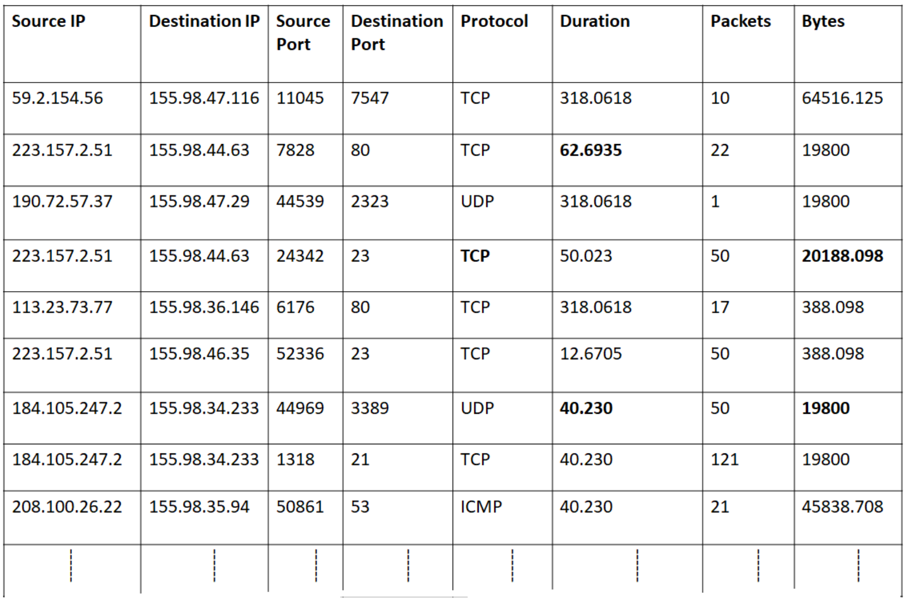
1) Replace missing values with the mean. For the features such as duration, total packets and bytes exchanged we assume that missing values are distributed similarly to the values that are present for each source IP. In this case, substituting values that represent the existing distribution, such as the mean, is a reasonable approach.

2) Replace missing values with the median/ mode. This is another justifiable way to handle missing-at-random data, although note that it gives a different answer. For categorical data, it's also common to use the mode, the most commonly occurring value. Missing protocols were handled using the mode approach, filling the missing values with the most occurring value for each source IP.

3) Delete columns that are missing too many values to be useful. If a feature is missing too many values, or if there is not enough information available to make reasonable assumptions about how to replace the missing values, you can delete the column entirely. If the feature does not provide useful information, including it can slow down the model's runtime. For many algorithms, having many noisy or uninformative columns can actually degrade their performance. In those cases, deleting these columns during data preparation is the best policy. 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 preceding page is one among them. After handling missing values our data looks as in **Table 3.2** on the next page.

3.1.2.2 Converting categorical to Numerical Variables

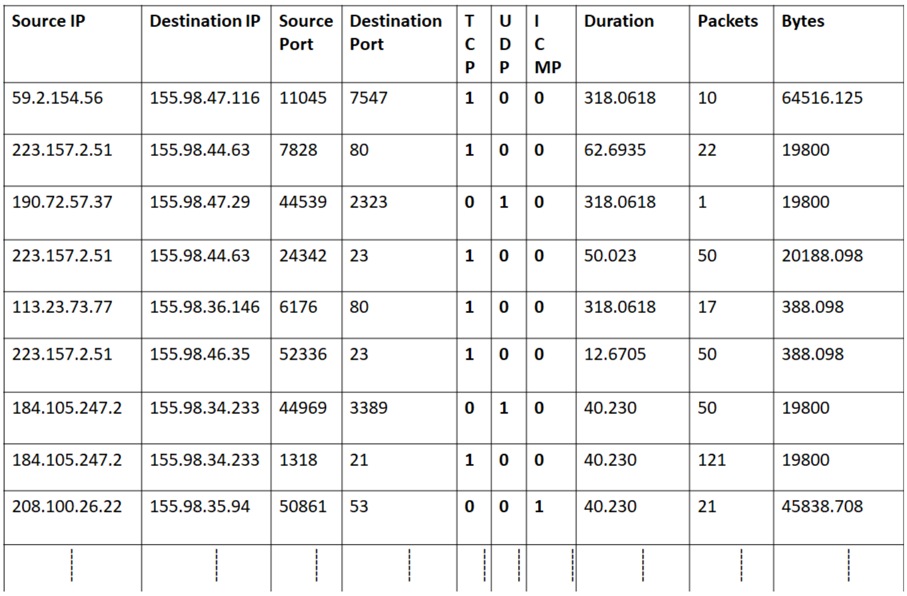
Features could be either numerical or categorical variables. When dealing with algorithms that work on numerical data we have to make sure that the categorical variables are handled. For example, in our case feature protocol, has values such as TCP, UDP and others. One approach is to encode them as 0, 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) minus 0 (TCP) doesn't mean



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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.2. After Handling Missing Data

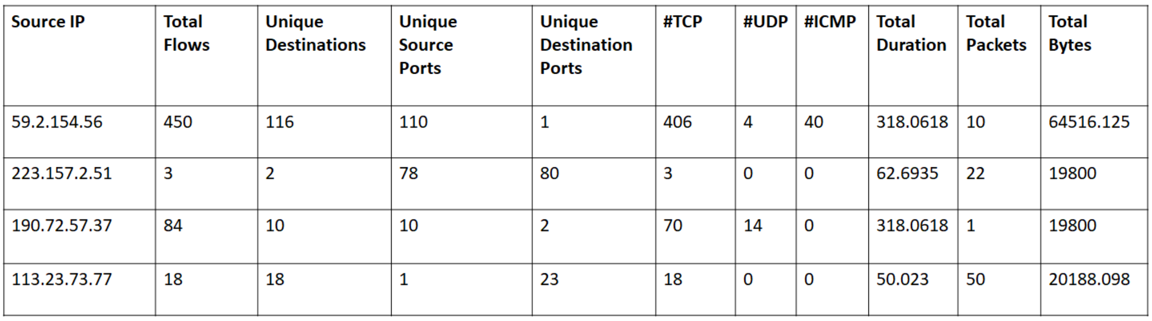
anything. Another approach would be to create a feature for each value of the categorical variable and mark it as 0 or 1 based on if the value is present or not. There are both pros and cons to this method. The pros are if we choose to aggregate the values on all features this conversion gives a numerical value to work with. The cons are scaling and standardization could be affected. **Table 3.3** on this page represents data set after this step.



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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.3. After Converting Categorical data to Numerical

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 current page shows a sample of aggregated data with ten features. The first record in the **Table 3.4** on this 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.



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

Table 3.4. Aggregated data by Source IP

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 current 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. **Table ??** on page ?? represents the normalized aggregated data set after applying scaling.

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 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

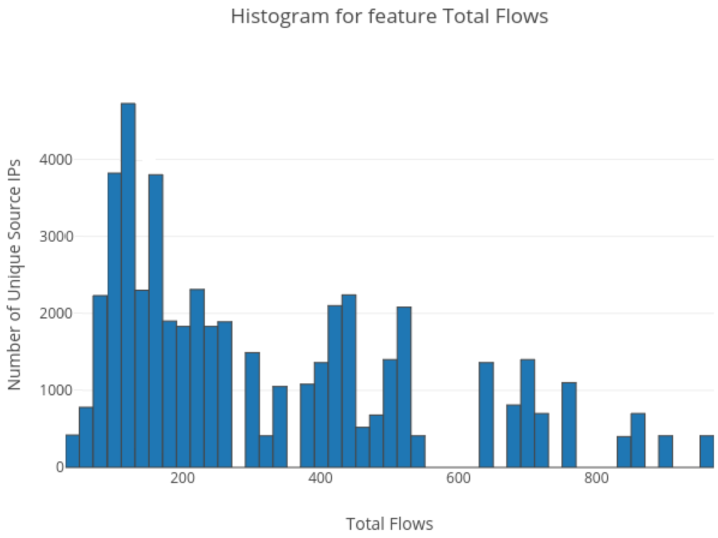


Figure 3.3. Skewed data before transformation

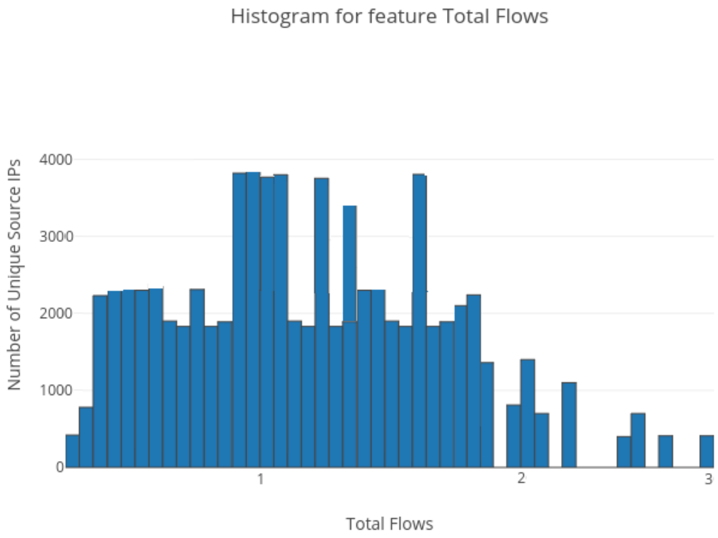


Figure 3.4. Normalized data

selection depend on text, numerical variables and they are three general classes of them Filter, Wrapper, Embedded methods. Selecting features in unsupervised learning scenarios is a much harder problem, due to the absence of class labels that would guide the search for relevant information. Problems of this kind have been rarely studied in the literature

[4] [6].

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 certain criteria[6]. it selects the features with the highest score. It is called the filter since it filters out the irrelevant features using given criteria. In wrapper methods, we try to use a subset of features and train a model using them. Based on the inferences that we draw from the previous model, we decide to add or remove features from your subset. Some common examples of wrapper methods are forward feature selection, backward feature elimination, recursive feature elimination, Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model. In backward elimination, we start with all the features and remove the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features. Recursive feature elimination, It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.

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 Host-Based Split

3.3 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.

Why K-Means ??

In the section 2.0.1 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($n^2 log n$) 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.

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).

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

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.

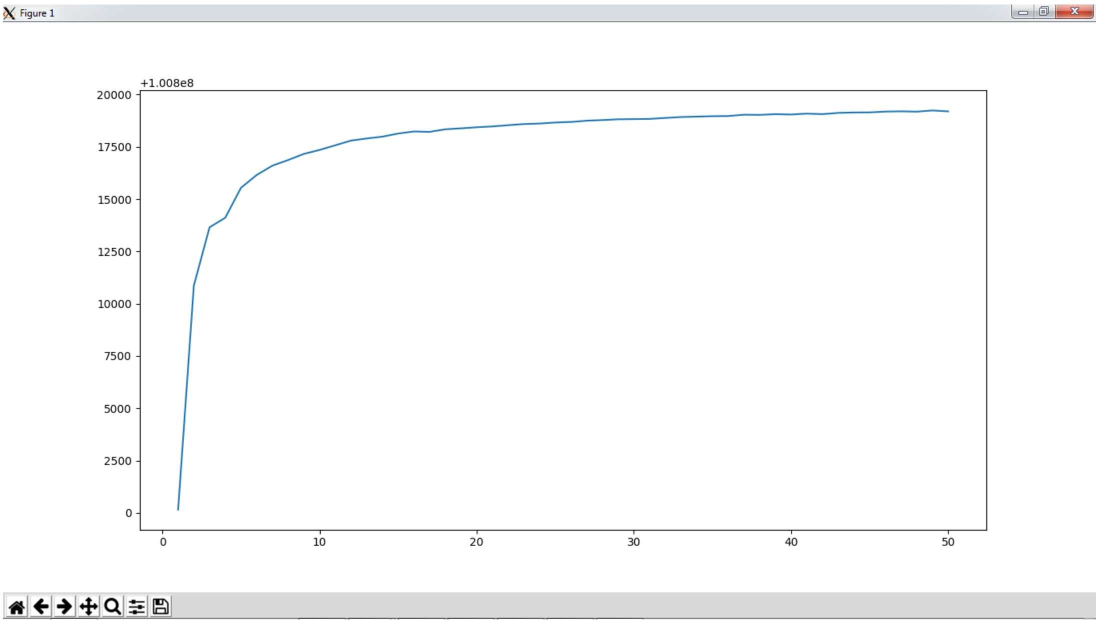


Figure 3.5. Elbow Method

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.

Cluster Labeling

We chose a reference day for labeling the clusters. Since the number of clusters are only seven we had manually inspected each cluster and labeled them based on the contents of

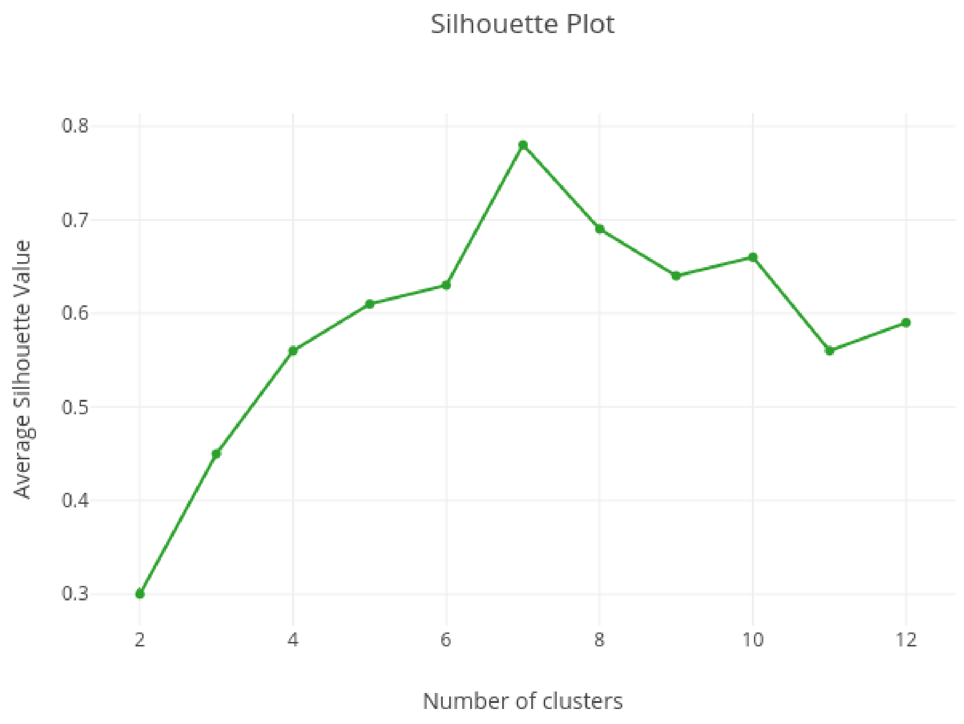
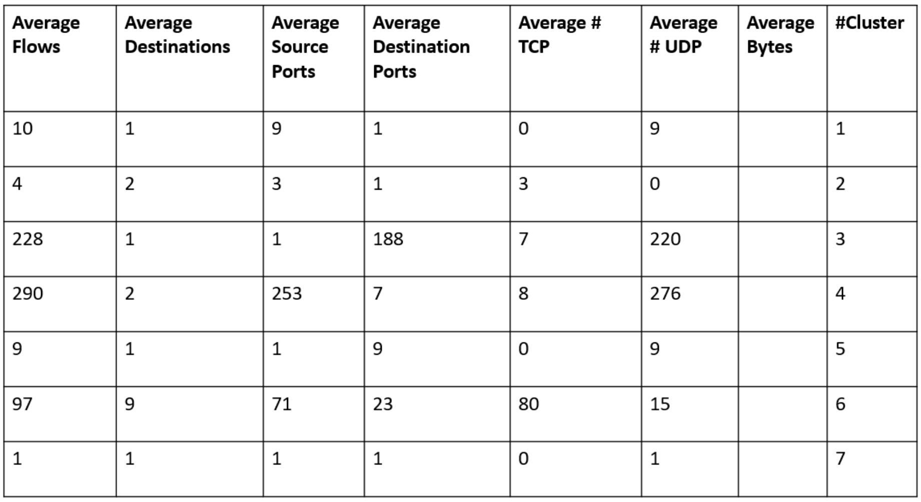


Figure 3.6. Silhouette Method

the cluster. **Table 3.5** on the next page shows clusters formed and the cluster centers. The first row describes that the cluster 1 has hosts which on an average sent 10 outgoing flows by using 9 different source ports targeting a single destination on the same port and all those flows were UDP flows.

We have mapped each of these clusters to human understandable classes which is shown in **Table 3.6** on page 23. The clusters represent the host behaviors as follows:

- Cluster 1 represents hosts which are sending DNS queries to our servers and they are classified as Low as they have an average of 10 flows.
- Cluster 2 hosts exchange normal tcp traffic. These hosts could various things ranging from exchanging messages to files.
- Cluster 4 represents the hosts similar to the Cluster 1 hosts which are sending DNS queries to our servers and they are classified as high as they have an average of 200 flows.



Average Flows	Average Destinations	Average Source Ports	Average Destination Ports	Average # TCP	Average # UDP	Average Bytes	#Cluster
10	1	9	1	0	9		1
4	2	3	1	3	0		2
228	1	1	188	7	220		3
290	2	253	7	8	276		4
9	1	1	9	0	9		5
97	9	71	23	80	15		6
1	1	1	1	0	1		7

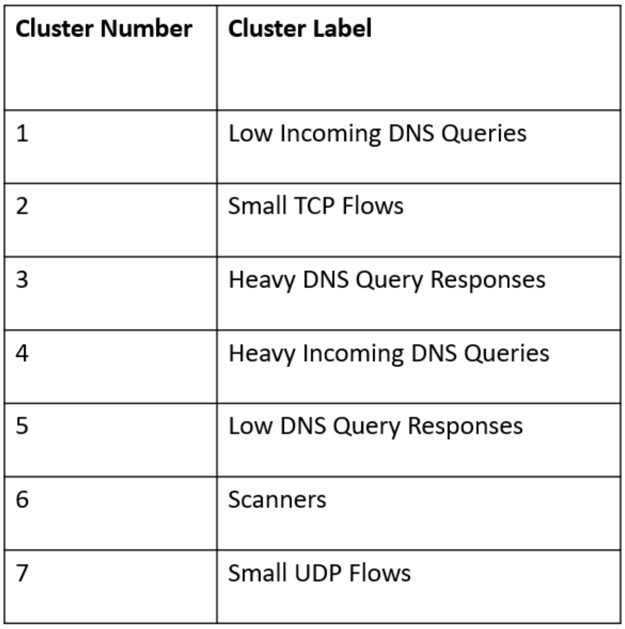
Table 3.5. Cluster Centers

- Cluster 3 and Cluster 5 represent to the Heavy and Low DNS Query Responses. The hosts in these clusters correspond to Heavy and Low DNS queries respectively.
- Cluster 7 represents only UDP flows.
- Cluster 6 represents anomalous hosts trying to scan through the network by sending heavy packets or trying to gain unauthorized access through open ports.

Cluster Comparison

As mentioned the overarching premise of our work is to analyze the behavior of hosts looking at the aggregated 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.

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 [16]. 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.



Cluster Number	Cluster Label
1	Low Incoming DNS Queries
2	Small TCP Flows
3	Heavy DNS Query Responses
4	Heavy Incoming DNS Queries
5	Low DNS Query Responses
6	Scanners
7	Small UDP Flows

Table 3.6. Cluster Labeling

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.

3.4 Applications

CHAPTER 4

IMPLEMENTATION

CHAPTER 5

RELATED WORK

In this chapter, we review works that relate to usage of Machine Learning/Data Mining 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 [5] 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 [5] concentration has been mainly on explaining the ML and DM methods where as Bhuyan et al. [3] in his paper described different techniques used for network anomaly detection in detail. Garcia's work [13] 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. [21] 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 [25] provides a perspective focused only on wireless network protection.

5.2 Machine Learning for Traffic Classification

Apart form using ML/DM techniques for intrusion detection one other field where these have been extensively used is to classify the network data. The survey [20] 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 [24] 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 [18]. This work was extended by applying Bayesian neural network approach [1] 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[2] 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 [2] 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. [10] 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 [2] . 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. [11] and Gao et al. [12] 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.[26] 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 [9] [8], Botnet Detectors [22] [17] [15]. Specifically, the botnet detector built by Livadas et al. [17] is of interest as they build a model using the aggregated flows similar to ours.

CHAPTER 6

RESULTS

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

[12] ——, *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.

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.