2014

Non-intrusive anomaly detection for encrypted networks

Luis C. Armendariz Jr.
West Virginia University

Non-Intrusive Anomaly Detection for Encrypted Networks

Luis C. Armendariz, Jr.

Thesis submitted
to the Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science in
Electrical Engineering

Roy S. Nutter, Jr., Ph.D., Chair
Katerina D. Goseva-Popstojanova, Ph.D.
Afzel Noore, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2013

Keywords: Intrusion Detection; Encryption; Clustering

ABSTRACT

Non-Intrusive Anomaly Detection for Encrypted Networks

Luis C. Armendariz, Jr.

The use of encryption is steadily increasing. Packet payloads that are encrypted are becoming increasingly difficult to analyze using IDSs. This investigation uses a new non-intrusive IDS approach to detect network intrusions using a K-Means clustering methodology. It was found that this approach was able to detect many intrusions for these datasets while maintaining the encrypted confidentiality of packet information. This work utilized the KDD '99 and NSL-KDD evaluation datasets for testing.

"We assumed the digital footprints we left behind
- our clickstream exhaust, so to speak -
were as ephemeral as a phone call,
fleeting, passing, unrecorded...

Our tracks through the digital sand are [in fact] eternal."

- Tom Zeller Jr.

ACKNOWLEDGEMENTS

I would like to take a moment to recognize those who have made my thesis, my college career, and all that I am blessed to have in life truly memorable. To my fiancée Mary-Leigh, whose faith, love, and many sacrifices have been an enduring beacon of hope throughout the years. To my parents whom have always been there for me and continuously supported and encouraged me throughout each of my college endeavors. To my nana whose stories and wisdom always inspired me during the tough times of college. To Mrs. Pierce and my church family who have supported and guided me and covered many of my college travel expenses. To Tom and his family who opened their home to me during the final stages of my college career.

I would like to take this time to recognize my advisor and mentor Dr. Roy S. Nutter, Jr. who gave me the opportunity to return to West Virginia University and pursue my Masters degree. His advice and dedication in and out of the classroom have truly been inspiring and have continued to motivate me in seeking higher excellences. To Dr. Noore, who taught me to always seek beyond where the bar was set. To Dr. Goseva-Popstojanova, who taught me the importance of confidentiality, integrity, availability, and authenticity of information.

Finally, I would like to recognize the Lane Department and West Virginia University for making the last 6.5 years truly some of the best moments of my life. Lastly, I'd like to recognize the WV State Police, the National White Collar Crime Center, the NCFTA, most notably Sergeant James E. Kozik, Jeremiah Johnson, Michael Shoukry, and each of my fellow graduate students.

Table of Contents

Chapter 1: Introduction

1.1 Introduction

One method of detecting an intrusion is to use an intrusion detection system (IDS). The IDS has the ability to monitor system or network traffic and compare it to a standard for analysis. Generally, a standard may be differentiated by an anomaly-based process utilizing behavior models or a signature-based process containing signatures of attack descriptions. In recent years, an effort to design and build systems that analyze network traffic over a variety of mediums has emerged. Most of the solutions utilize a type of anomaly or signature based approach that analyzes packet payloads for additional data.

The issue with these systems is that they are unable to analyze network traffic that is sent over an encrypted channel, due to the payload of the packets being inaccessible. Only a limited quantity of IDS systems are designed to handle encrypted information. Many of these systems require protocol modifications or special infrastructures because of abnormally high false alarm rates. This paper will investigate a methodology to analyze encrypted network traffic with a K-Means clustering algorithm. Additionally, it will utilize TCP traffic properties to evaluate the possibility of an intrusion. An intrusion detection system (IDS) can thus monitor network traffic for potential intrusions while maintaining the confidentiality of the packet information.

In order to validate the effectiveness of detecting intrusions with the proposed methodology, the investigation will implement the proposed system using the KDD '99 and NSL-KDD evaluation datasets. Furthermore, the results of the evaluation will be compared against additional methodologies from similar research experiments. By doing this, the effectiveness of this strategy can be analyzed.

1.2 Statement of Problem

Intrusion Detection systems typically utilize either a signature-based or anomaly-based approach to analyze in-the-clear network traffic connected to a network or host machine. However, encrypted communications deny use of payload-related data. One approach to analyze encrypted networks is non-intrusive. A non-intrusive approach will be examined to determine if utilizing a K-Means clustering model and TCP traffic properties will provide high precision and recall with a lower false rate than the only other known existing non-intrusive methodology. [1,49] This will be accomplished by validating the effectiveness of the algorithm using the KDD '99 and NSL-KDD evaluation datasets.

1.3 Organization

This paper is separated into six chapters. The first chapter provides an introduction to intrusion detection and the initial problem statement. The second chapter provides a more detailed background to intrusion detection as well the current research in this field of study. The third chapter discusses the proposed intrusion detection system and the evaluation of the proposed system. The fourth chapter examines the data collected to either support or refute the initial theory, as well as discusses the final conclusions and where this work can be expanded in the future. Finally, the fifth and sixth chapters list the bibliography and additional appendices.

Chapter 2: Literary Review

This chapter provides a review of the current research in the area of intrusion detection. The first section discusses the background and evolution of intrusion detection to its current methodologies. The second section provides a brief review of encryption and the current research related to detecting intrusions over encrypted channels. The third section discusses clustering methodologies. Finally, the fourth section discusses goals for next generation Intrusion Detection Systems (IDSs).

2.1 Intrusion Detection

2.1.1 Specification

Intrusion Detection is the process of monitoring a network or system for potential signs of malicious activities or policy violations. Possible types of incidents include violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices. [2].

2.1.2 Background

In the beginning, intrusion detection was first performed by system administrators. The process consisted of sitting in front of a user console and monitoring user activities for potential intrusions. Although this methodology was effective at the time, this form of detection was very ad hoc and did not allow for scalability. [22]

However, the awareness of intrusion detection began to spread when a United States Air Force (USAF) paper written by James P. Anderson was published in 1972. This paper identified how the USAF had "become increasingly aware of computer security problems" and spurred people to begin asking questions on how to safely secure information without compromising security. [16][17] From this study, James P. Anderson published a paper in 1980 called *How to use accounting audit files to detect unauthorized access*. [18] This paper outlined many ways to improve computer security auditing and surveillance at customer sites, as well as paved a way for misuse detection in mainframe systems [16].

From here, Dorothy Denning and Peter Neumann developed the first model of a real-time IDS called the Intrusion Detection Expert System (IDES) between 1984 and 1986. [19] Initially, this system was a rule-based system, trained to detect known malicious activity. However, the system was

refined to incorporate the statistical analysis of user profiles, becoming the Next-Generation Intrusion Detection Expert System (NIDES) [20]. Over time, the system was further enhanced to become the Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) project. [21]

Throughout the 1980s and 1990s, much of the intrusion detection research was based on these initial models and designs. Many projects thrived due to funding from the United States (U.S.) government. Projects such as Discovery, Haystack, Multics Intrusion Detection and Alerting System (MIDAS), Network Audit Director and Intrusion Reporter (NADIR), Netranger, RealSecure, and Snort were developed during this period [16].

2.1.3 Detection Techniques

Today, there are two types of detection techniques that are commonly used to design Intrusion Detection Systems: Misuse Detection (MD-IDS) and Anomaly Detection (AD-IDS). [1]

2.1.3.1 Classifications

It is important to note though that no single IDS is perfect. [2] Each IDS faces unique problems and challenges, primarily due to the fact that network traffic continues to increase in complexity. [2] As such, erroneous results produced by an IDS are divided into two categories: false positives and false negatives. [2]

	Alert not generated	Alert is generated
Passive Activity	True Negative	False Positive
Intrusive Activity	False Negative	True Positive

Table 1: False Positives and False Negatives

A false positive is a sequence of innocuous events that an IDS erroneously classifies as intrusive. [2][15] However, a false negative is a sequence of unwanted traffic or intrusion attempts that an IDS fails to detect or report. [2][15]

Above all, the reduction of both false positives and false negatives is a critical component in intrusion detection. [15] Both create problems for system and security administrators and may require additional system calibration. [2] However, while false positives can create the burden of sifting through cumbersome amounts of data, they are generally more acceptable than false negatives. [2] This is because false negatives, as undetected, do not provide a security administrator with the opportunity to review the data. [2]

2.1.3.2 Misuse Detection

The first technique, known as Misuse Detection, focuses on identifying intrusions using a predetermined knowledge base [15]. Usually, this is done by utilizing a signature-based approach in order to search for well-known attack patterns. [1]

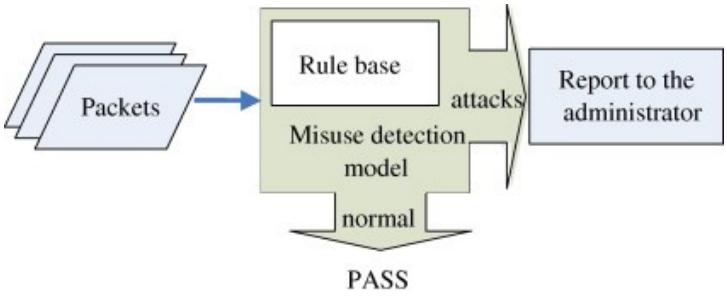

Figure 1: Misuse Detection Model [24][25]

Each attack signature, or fingerprint, is compared with the current system activities in order to find strange or abusive use of a network or system.

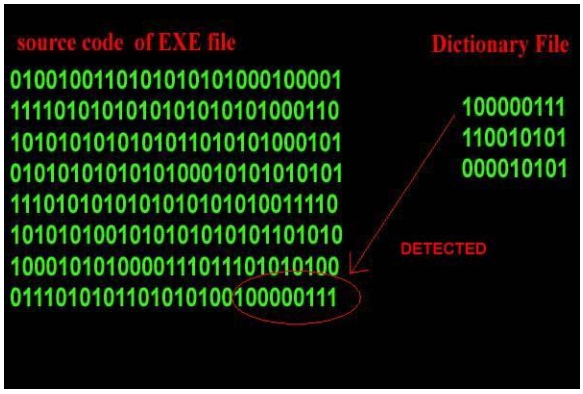

Figure 2: Detecting a Signature [26]

As such, MD-IDSs have shown to have quick detection speeds and manageable configurations [2], as well as produce a low number of false positives [15]. Commercial implementations have also seen widespread adoption and success. [16][20]

However, signature-based systems and approaches are reactive by nature [10][11] and thus restricted to recognizing only known attacks. Additionally, an attacker has the ability to modify an attack, rendering it undetectable by a MD-IDS. [2]

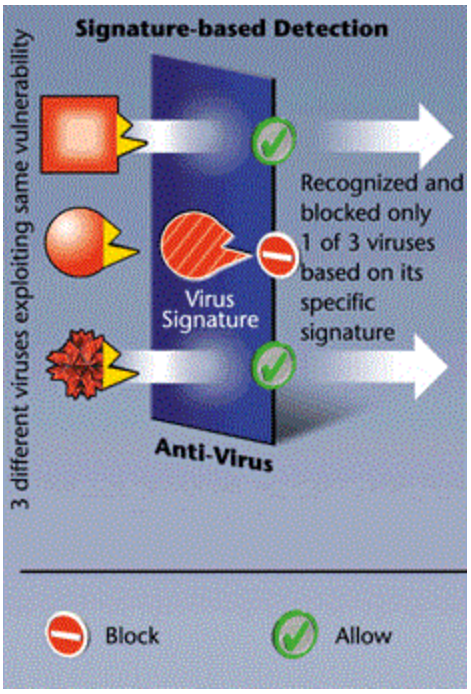

Figure 3: Detection of Signature Modifications [23]

As a result, MD-IDSs often produce a high number of false negatives, and their efficiency is dependent upon continuous updates and response times. [1]

2.1.3.3 Anomaly Detection

The second technique is known as Anomaly Detection. This process utilizes a heuristic-based approach, in order to build a model of the "normal" system or network behavior. [1] Using this model, or profile, intrusions can be detected as anomalies, or deviations, from the expected behavior of the system. [1]

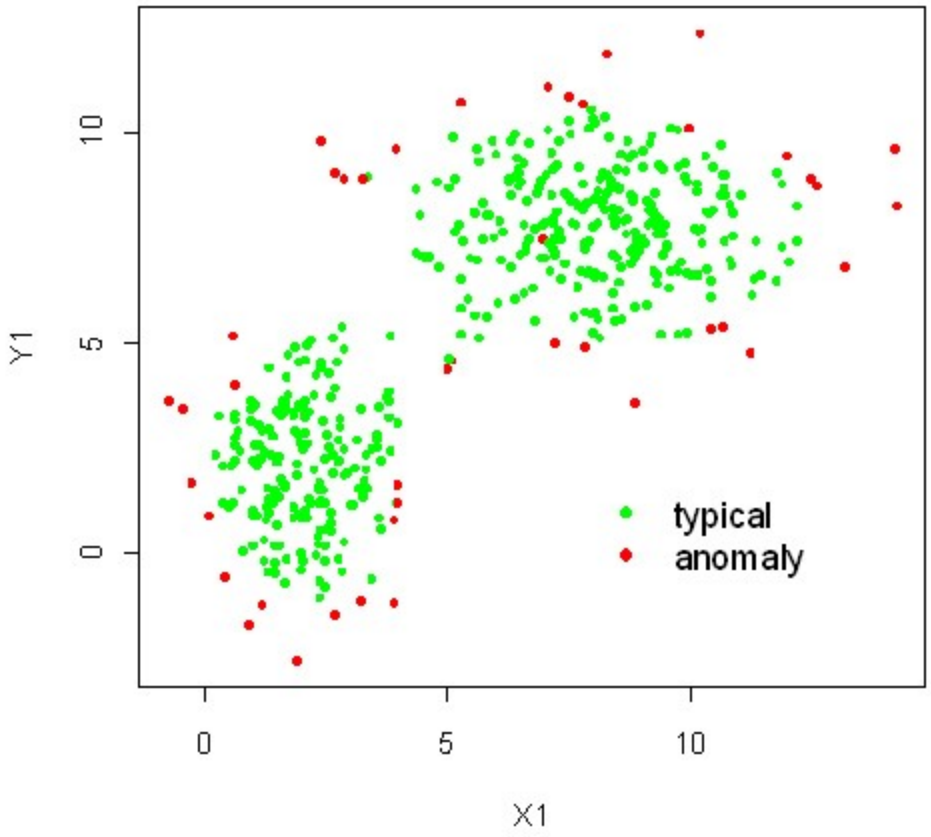

Figure 4: Anomaly Detection [27]

The primary advantage of this approach is that AD-IDSs are able to detect new and yet unknown threats. [1] This provides additional support that signature-based systems are unable to supply.

Figure 5: New and Unknown Threats [28]

As such, a wide variety of methods have been explored in order to approach the anomaly detection issue, including neural networks, artificial intelligence, data mining algorithms, genetic algorithms, and statistical models. [15]

However, AD-IDSs are prone to an increased amount of false positives, since a variety of factors can contribute to producing an anomalous behavior (e.g. implementation errors). [15] Furthermore, the allocation of a training phase to develop the analysis model may also be required, depending upon the approach.

2.1.4 Implementations

The placement of an Intrusion Detection System (IDS) is another vital aspect of the system's effectiveness. [1] Depending on whether the system is designed to monitor traffic to a single host or a network of devices, can also determine what types of intrusions the system may encounter. As such, many IDSs today often utilize one of two common implementation approaches.

2.1.4.1 Host-based Intrusion Detection System

The first approach uses a host centric design, where an IDS requires a small program, or agent, to be installed on a single device or machine. This type of implementation is known as a Host-based Intrusion Detection System (HIDS). [2]

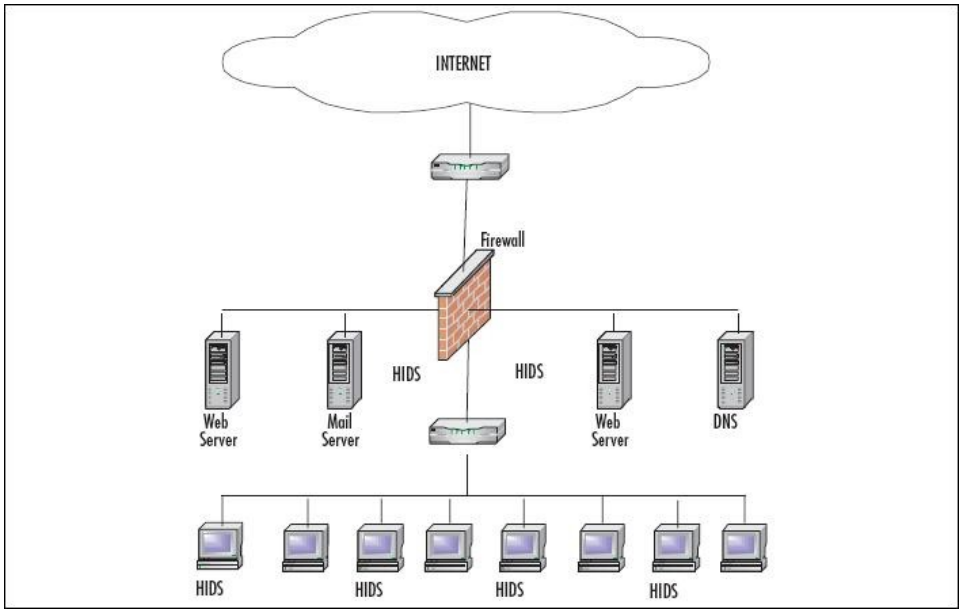

Figure 6: Host-based Intrusion Detection System Setup [31]

One primary advantage of this implementation is that it allows a HIDS to access and analyze system-specific settings and information in order to detect intrusions. [1][2] This includes the local security policy [2], the file system, network events, system calls [15], system commands, system logs, and security logs. [14] Thus, it is important for a HIDS to appropriately choose the system characteristics it will monitor. [15]

However, the primary concern with this implementation is that a HIDS agent must be installed on the machine it intends to analyze. [2] As such, configuration settings must be specific to that machine, operating system, and software [2], limiting the HIDS in scalability and increasing the complexity of system management.

2.1.4.2 Network-based Intrusion Detection System

The second approach utilizes a network centric design, where an IDS is placed on a network to monitor information that is passed between multiple hosts or to a unique device. This type of implementation is known as a Network-based Intrusion Detection System (NIDS).

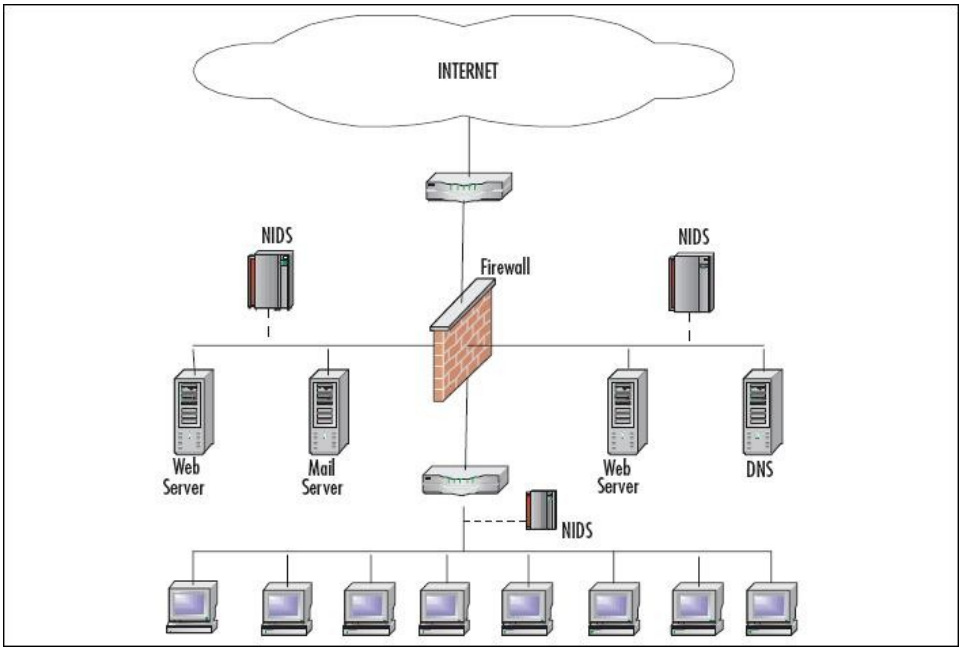

Figure 7: Network-based Intrusion Detection System Setup [31]

Usually, this setup consists of a network application, or sensor, integrated together with a Network Interface Card (NIC). The NIC operates in promiscuous mode in order to collect the traffic or data, while a separate interface is used for management of the system.

The primary advantage of a NIDS setup is that it is able to analyze traffic at all layers of the Open Systems Interconnection (OSI) model [2] and detect attacks against the network as a whole. [1] A few examples of this would be distributed denial-of-service (DDOS) attacks, policy violations, and various classes of malware. [14]

However, a NIDS is limited to the data that is made available by the network. As such, unique system information such as audit logs is often restricted or unavailable for analysis.

2.1.5 Current Research

2.1.5.1 Threats

Today, many well-known attacks and new types of threats can be detected and impeded from causing harm. [1] However, all systems suffer from security vulnerabilities and are subject to electronic attacks.

With the commercial success of the Internet and the ability to carry out attacks from afar, e-Crime has evolved into a multi-billion dollar market. [1,32]. Malicious code has now proliferated through the development and distribution of attack kits, requiring no in-depth technical knowledge. [1] Each professional kit contributes to the growing numbers of new signatures with each set of new code. [1]

As the world continues to connect with the cyberworld, attackers are becoming more sophisticated using automated tools to penetrate systems and organizing highly coordinated and intricate attacks. [2] As such, targeted attacks have also increased [32-33], including the most recent commercial attacks on Nintendo[3-4], Ubisoft[5], and Sony [6-9]. Small businesses have become more viable targets, accounting for fifty percent of all targeted attacks in 2012. [33] Furthermore, the number of cyber espionage attacks on military targets has also increased, such as the Flamer and Gauss worms [33] and recent Social Engineering exploits [34].

The dissemination of malicious software and packages is not restricted to networks anymore. Flash drives used as promotional prizes have become popular instruments for Trojans. [1] Once a device is connected to a computer, the Trojan installs itself onto the host system or network, bypassing the security systems. Additionally, formerly secure systems such as Supervisory Control And Data Acquisition Systems (SCADA) can be compromised with the use of offline-propagation. [1] Thus, protecting a system from only external threats is not enough.

With the evolution of technology and the interconnectivity provided by the Internet, additional problems continue to arise. One important note is the shift from attacks directed at operating systems or network protocols to attacks utilizing vulnerabilities in the application layer. [1] As new applications are created and developed, the potential for flawed program

code to enter the market continues to increase. As such, the quantity of Zero-Day vulnerabilities has increased in recent years. [1]

While vendors continue to supply consumers with patches and updates to discovered vulnerabilities, these updates may sometimes be delayed utilizing a fixed patch-day policy. [1] Even more so, many users may not take the appropriate safety precautions, due to often strenuous, complex, or often changing security configurations. As such, the most successful exploits are often utilizing vulnerabilities that were reported more than a year ago. [1]

2.1.5.2 Increased Security Measures

Based on these recent changes, more and more services are beginning to offer protected access, such as Transport Layer Security (TLS). This trend will continue to grow, through the use of IPv6 IPSec [1].

However, many available systems will be unable to cope or scale to challenges like encryption, since it denies the use of payload data for evaluation. [1]

As IDS technologies continue to evolve, encryption will continue to become a crucial factor to train the application of IDSs. [1,2]

2.2. Encryption Analysis

2.2.1 Encryption with Intrusion Detection

Today, encryption is becoming a more common component and entity in data communications. Protocols such as TLS and IPSec offer services that help in safeguarding private information. However, as a result, common IDS structures and systems are unable to analyze packet payloads in order to detect intrusions. As such, new methodologies have been examined in order to integrate common intrusion detection designs with encrypted networks.

2.2.2 Methodologies

2.2.2.1 Protocol-based

The first methodology is a protocol-based approach, where malicious activity is detected based on misuse of the encryption protocol. [1]

The primary advantage of this approach is that common attacks based on vulnerabilities in the protocol can be analyzed and detected. However, this approach is limited mainly to the misuse of the encryption protocol. As such, additional attack vectors such as application-level attacks (e.g. SQL injection, buffer overflow, cross-site request forgery) are usually not detected since the payload of the packets is not decrypted and analyzed. [1]

For example, Joglekar et al. [46] developed an anomaly-based IDS for detecting malicious use of cryptographic and application-level protocols. This system, denoted as *ProtoMon*, instrumented shared libraries for these protocols in order to detect intrusions. As the monitoring was integrated into the protocol handling, attacks on the encryption protocol were able to be detected. However, malicious activities hidden within the encrypted channel remained undetected.

2.2.2.2 Intrusive

The next methodology is an intrusive approach, where modifications are implemented onto the network architecture or the encryption protocol. [1]

The main advantage of this approach is that it provides a way to perform deep packet inspection of payloads while maintaining the confidentiality of the information. However, this approach strongly depends on modifications

to the protocols and the network infrastructure. As such, this creates heavy management overhead and is often limited in scalability.

For example, Goh et al. [47-48] proposed and developed a detection framework that allowed a NIDS to analyze network traffic without compromising the confidentiality of a VPN. This approach was able to detect application-level attacks and evasion attacks; however, it was limited in scalability due to increased network overhead and implementation challenges.

2.2.2.3 Non-Intrusive

The last methodology is a non-intrusive approach, where statistical models and analysis methods are applied to encrypted traffic. [1]

The primary advantage with this approach is that it provides a way to analyze network traffic without relying on packet payloads. Furthermore, it maintains the confidentially of the information without modifications to the network infrastructure. However, application-level attacks may not be detected since the payload is not decrypted and analyzed. [1] Additionally, this strategy also has the potential to have a high false positive rate, decreasing its suitability to an online implementation.

In one approach, Foroushani et al. [49] proposed a system to detect anomalous behaviors in SSH2 encrypted accesses, using intrusion signatures generated from traffic information (e.g. access frequency, TCP traffic specifications). This system was able to detect a variety of attacks with high accuracy; however, it was not suitable to online implementation due to a high false alarm rate.

2.3 Clustering Analysis

In this section, an introduction is given to common clustering methodologies and approaches. The first area gives an overview of clustering and a few of its main advantages for data analysis. The second area discusses clustering models and common algorithms and approaches that have been utilized in research. The third area discusses common clustering techniques that have been used to classify data. The last section discusses current clustering research specific to the K-Means algorithm.

2.3.1 Clustering Overview

Clustering is the process of organizing a set of objects in such a way that information in the same group, or cluster, is more similar than to information in other groups, or clusters. [35]

It has a vast history, expanding across disciplines such as biology, psychology, geology, and marketing. [36] Today, this process of analyzing data and information is found in many fields, including machine learning, pattern recognition, and information retrieval. [36]

One of the main advantages of clustering is that it allows the analysis of cluster groups containing similar objects rather than the analysis of each individual object from a respective data set. However, the notion of a "cluster" cannot be precisely defined, as the process of clustering is the general task to be solved. [35] As such, there are many different types of clustering models, with each model employing a different inductive principle for the definition of a cluster. Furthermore, each induction principle contains many clustering algorithms that may be used for data analysis. [35]

2.3.2 Clustering Models

A clustering model is the basic structure that is used to represent a cluster, or a group of data. This model usually relies on an inductive principle, in order to define a cluster, as well as a clustering criterion to select a "best fit" structure given a set of data. [35] By defining what the clustering criterion is, it also aids in accounting for the similarity between the cluster groups. [35]

For example, one commonly implemented clustering model is the probabilistic model. This may utilize an inductive principle such as the Maximum Likelihood (ML) approach, which states to choose the model that maximizes the probability of the data being generated by such a model. [35] The clustering criterion would be the mathematical expression of the inductive principle, while the clustering algorithm would implement this criterion such as the Expectation Maximisation method. [35]

However, it is also common for the names of clustering models to be used interchangeably with the inductive principle. As such, it has become difficult to identify if the clustering model name refers more to the model or the induction principle. [35]

A few common clustering methods are hierarchical clustering, partitional clustering, distribution clustering, and density clustering.

2.3.2.1 Hierarchical Clustering

Hierarchical Clustering, or connectivity models, builds a model based on the distance between connected objects. The core inductive principle is based on the guideline that objects are more related to nearby objects than to far away objects. As such, algorithms that utilize this model develop clusters based on object distances and represent clusters using a dendrogram, or tree diagram.

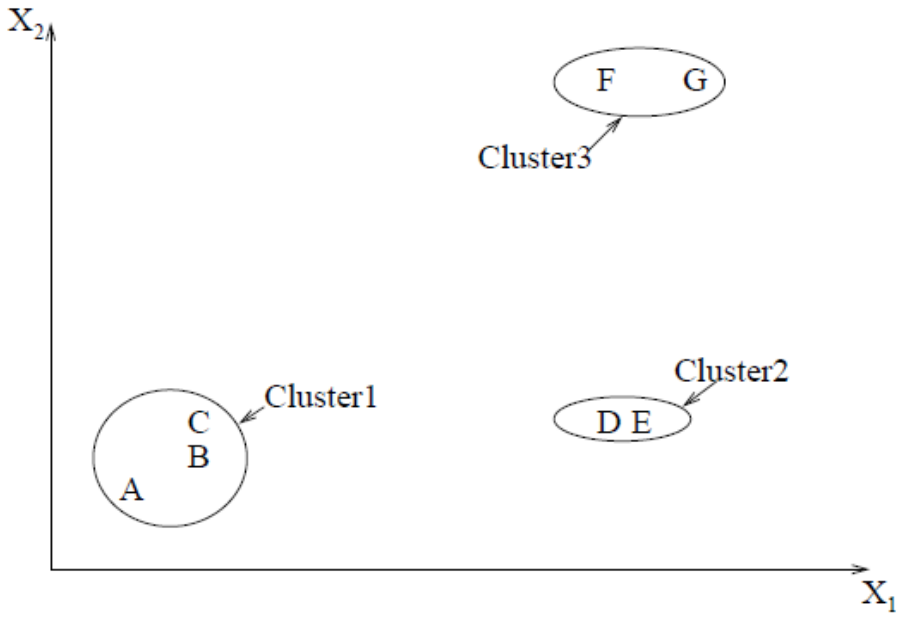

Figure 8: Hierarchical clustering points within 3 clusters [36]

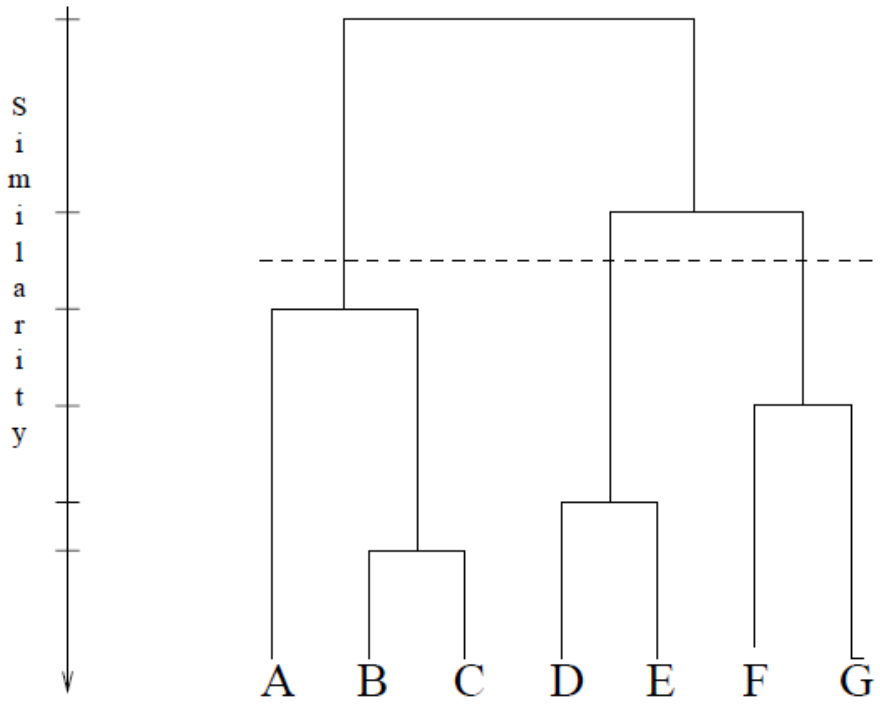

Figure 9: Hierarchical clustering dendrogram from single-link algorithm [36]

However, each algorithm differs by how the distances for each object are computed. A few common algorithms for connectivity models are the Linkage Clustering algorithms, including single-link, complete-link, and average link clustering. Of these algorithms, single-link clustering and complete-link clustering are most common. [36]

In single-link clustering, the distance between two clusters is the minimum of the distances between all pairs of patterns drawn from the two clusters. [36] However, in complete-link clustering, the distance is the maximum of all pairs of patterns derived from the two clusters. [36] As such, complete-link clustering produces tightly bound clusters, while single-link clustering suffers from a chaining effect. In contrast, single-link clustering is more versatile than complete-link clustering, even though noisy patterns may develop. [36]

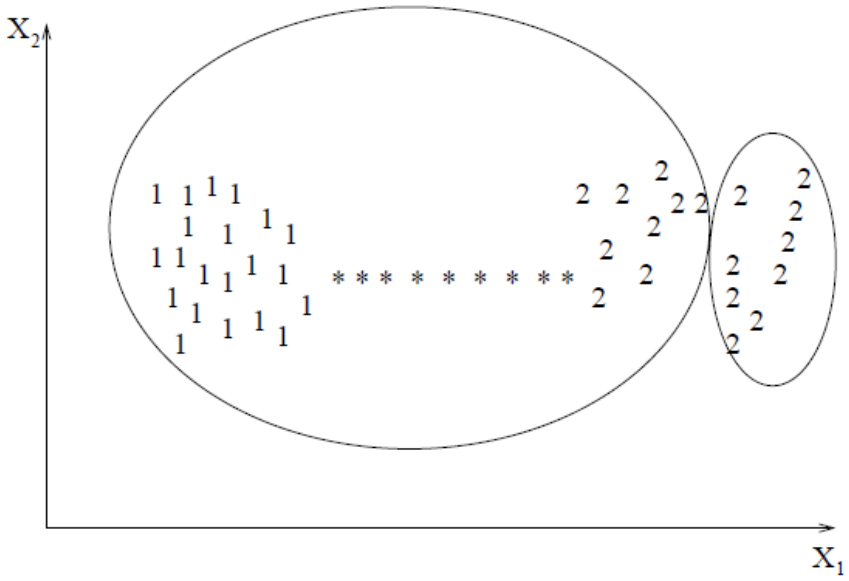

Figure 10: Single-link clustering with 2 data sets connected by noisy patterns [36]

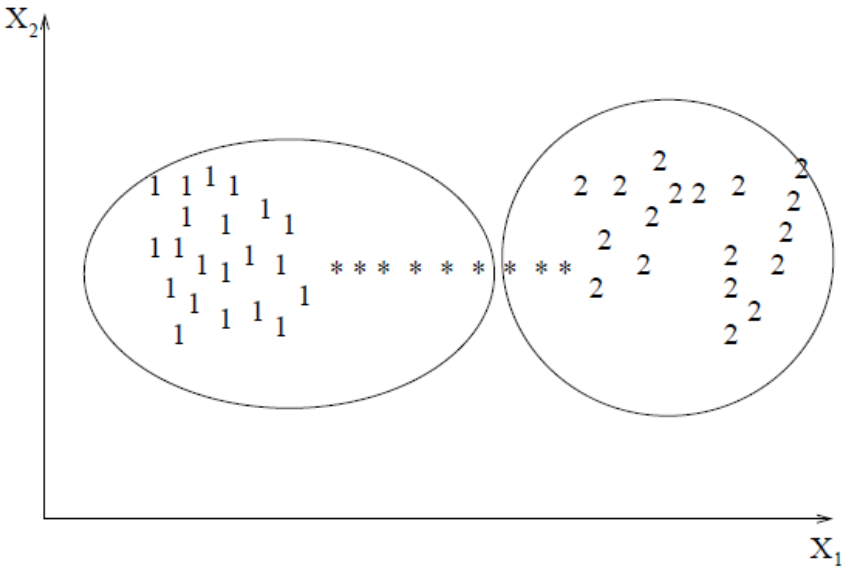

Figure 11: Complete-link clustering with 2 data sets connected by noisy patterns [36]

While this clustering methodology is useful in organizing a set of objects into a dendrogram, it does have its disadvantages. First, most hierarchical algorithms' results are not produced as a unique partitioning of a data set but as a hierarchy. As such, a user would still need to choose the appropriate clusters from the results. Second, this methodology is not

robust against outliers and can cause either additional clusters to form or other clusters to merge (i.e. Chaining Phenomenon). [36] Lastly, the complexity for most connectivity model algorithms is usually on the scale of $O(n^3)$. Even optimized methods for these algorithms are on a scale of $O(n^2)$. As such, many of these algorithms are too slow for large data sets. [36]

2.3.2.2 Partitional Clustering

In partitional clustering, or centroid models, a single partition of the data is obtained instead of utilizing a clustering structure. [36] The primary inductive principle in this approach is that a cluster is represented by a central vector, which may or may not be a member of the original data set.

A few algorithms that are common in this approach are the squared error algorithms and the graph-based algorithms. [36]

2.3.2.2.1 Squared Error Algorithms

The squared error algorithms utilize one of the most frequently used criterion functions in partitional clustering. This technique is known as the squared error criterion.

In this methodology, an initial partition is selected with a fixed number of clusters and cluster centers. Each data object or pattern is assigned to a cluster center, while cluster centers are continuously recomputed. Clusters are continuously split and merged again using heuristics information [36], until the algorithm converges or cannot be improved.

The primary goal of this operation is to obtain a partition for a fixed number of clusters that minimizes the squared-error value, or the objective function. [39,40] This value is the sum of the distances between each individual object or pattern and its cluster center. [39] An example of the clustering equation may be found in the figure below.

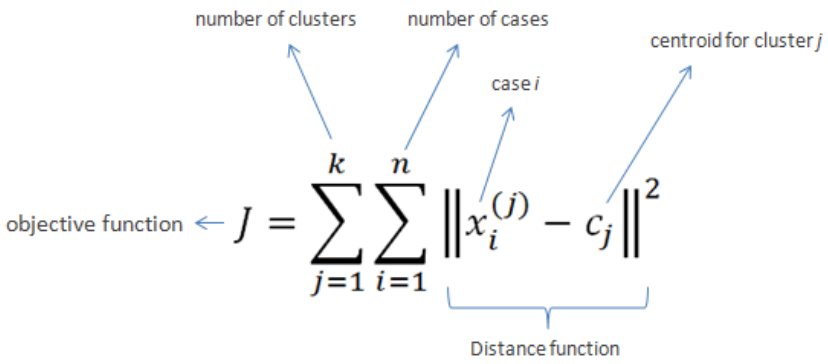

Figure 12: Example of Squared Error Criterion Equation [40]

For a clustering of a pattern set (χ) containing K clusters, the value $X_i^{(j)}$ is the i^{th} pattern belonging to the j^{th} cluster and c_j is the centroid of the j^{th} cluster.

The squared error criterion function is often effective with isolated and compact clusters. Furthermore, the Euclidean distance function is commonly used in conjunction with this approach. One of the most commonly used squared error algorithms is the K-Means clustering algorithm. [36]

2.3.2.2.2 K-Means Clustering

For the K-Means algorithm (i.e. Lloyd's algorithm), the methodology builds upon the basic principles of squared error algorithms. First, a random initial partition is selected with *n* objects and *k* cluster centers. Then, each data object is assigned to the closest cluster center, or the cluster with the nearest mean. Once this is accomplished, each cluster center is recomputed and a new set of *k* cluster centers are utilized to reassign objects. Convergence is achieved when no or minimal objects are reassigned to new cluster centers or when the squared error value ceases to decrease significantly. [36,40]

The K-Means algorithm is used quite often due to its implementation ease and its O(n) time complexity. However, one disadvantage with this technique is that it is sensitive to the selection of the initial

partition and may converge at a local minimum of the criterion function value instead of the global minimum. [36]

There are also several variations to the K-Means algorithm strategy. For example, some implementations allow the separation and aggregation of resulting clusters in order to select a more optimal partition. [36] Other variations may utilize different criterion functions in order to optimize the results, such as the dynamic clustering algorithm. [36]

Additional discussions for variations, modifications, and optimizations to the K-Means algorithm may be found in Section 2.3.3 Clustering Research.

2.3.2.2.2 Graph-Theoretic Clustering

In graph-based algorithms, the main approach is to construct a minimal spanning tree (MST) for the data. Using this tree, the MST edges with the largest lengths may be removed one-by-one, in order to generate the appropriate clusters. [36] An example of this process may be found in the figure below.

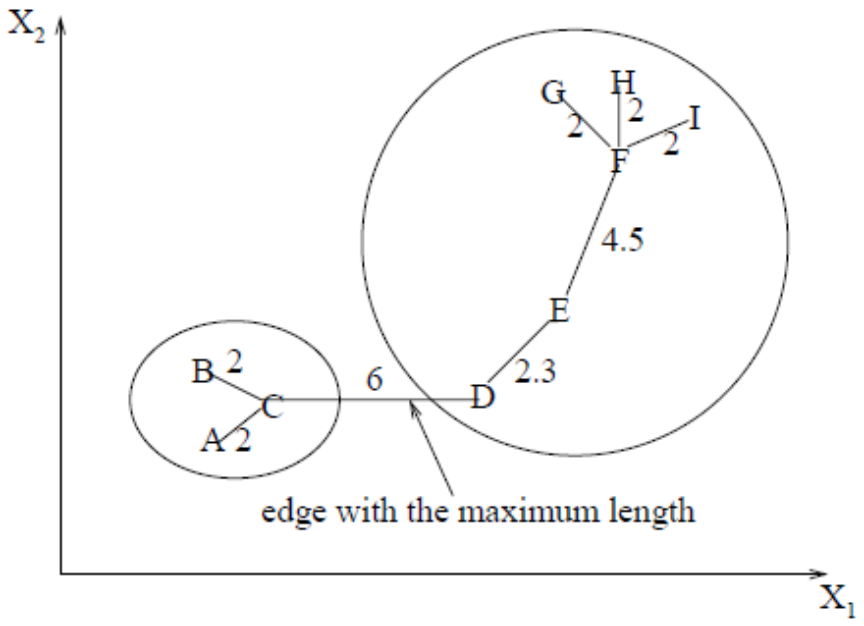

Figure 13: Graph-Theoretic Clustering MST [36]

One interesting point to note is that hierarchical clustering approaches are also related to graph-theoretic clustering models. For example, single-link clusters are subgraphs of the MST, while complete-link clusters are maximal complete subgraphs, considered the strictest definition of a cluster. [36]

2.3.2.3 Distribution Clustering

In Distribution Clustering, or distribution models, the primary inductive principle is that the patterns to be clustered are part of one or more distributions. Thus, the goal is to identify each of the input parameters and their corresponding number. [36] As such, this methodology is closely related to statistical approaches.

The most prominent algorithm used in this approach is the Expectation-Maximization (EM) algorithm, also referred to as EM-Clustering.

2.3.2.3.1 EM Clustering

In EM-Clustering, the data set is often modeled using a fixed number of Gaussian distributions. Each distribution is randomly initialized using a set of mixing parameters. With each iterative pass, the parameters are rescored and optimized in order to better match the data set. [36]

Overall, this methodology produces a set of clusters and complex models for these clusters that are able to capture correlation and dependence attributes. However, one key issue with this strategy is that it may suffer from overfitting, or when the statistical model may describe a random error or noise instead of the underlying relationship. Additionally, many real data sets may not have a mathematical model available for the algorithm to optimize. Thus, an additional burden is placed on the user to choose the appropriate data models for the analysis.

2.3.2.4 Density Clustering

In Density Clustering, or density models, the core inductive principle defines a cluster as a region in a data set with higher object density. Objects

that may be found or located near sparse areas are defined as noise or cluster border points. [37] A few algorithms that are common in this approach are the DBSCAN, OPTICS, and the DeLi-Clu algorithms.

The DBSCAN algorithm, or density-based spatial clustering of applications with noise, is one of the most commonly used density-clustering methods and utilizes the notion of density-reachability. Likewise, the OPTICS algorithm, or ordering points to identify the clustering structure, may be viewed as a generalization of the DBSCAN algorithm to multiple ranges. While each algorithm offers advantages similar to those of the Linkage Clustering algorithms, the primary drawback is that the algorithms expect a type of density drop in order to detect cluster borders. As such, they are not able to detect intrinsic cluster structures which are commonly prevalent in real-time data sets.

The DeLi-Clu, or Density Link-Clustering, algorithm, merges ideas found in both OPTICS and Single-Link clustering. This reduces the required amount of defined parameters and offers performance improvements over OPTICS by using an R-tree index.

2.3.3 Clustering Techniques

Similar to clustering models, there are also a variety of clustering techniques that can be applied to various clustering algorithms. These techniques may aid in defining the guidelines for the algorithm or clustering criterion.

2.3.3.1 Agglomerative vs Divisive

This technique relates to the algorithmic structure and operation. In an agglomerative approach, each pattern is defined within a distinct cluster and continuously merged until a specified stopping criterion is reached. However, in a divisive approach, all patterns are defined within a single cluster and continuously split until a specified stopping criterion is satisfied. [36]

2.3.3.1 Hard vs Soft Clustering

This technique relates to the how the algorithm places objects or patterns within defined clusters. A hard clustering approach places each data object

within a single cluster, during operation and output. However, a soft clustering (i.e. fuzzy clustering) approach assigns each data object to a set of clusters using relative degrees of memberships. [36]

2.3.4 Clustering Research

Recently, there have been many new research developments with regards to improvements or modifications to the K-Means clustering strategy. Many of these variations offer additional insight as to how to improve the accuracy of the detection algorithm and reduce the computational expenses.

For example, Xiao et. al. [38,50] proposed an approach where the K-Means algorithm was associated with the Particle Swarm Optimization (PSO) algorithm. This approach, denoted as *PSO-KM*, produced results that maintained a high accuracy when detecting probe attacks, denial-of-service (DoS) attacks, and user-to-root (U2R) attacks. It was also effective in converging towards a global optimum rather than a K-Means local optimum. However, the approach produced low results in detecting root-to-local (R2L) attacks and was unable to overcome the K-Means dependency on the number of clusters.

In another study, Gaddam et al. [41] proposed a supervised anomaly detection approach, known as *K-Means+ID3*, which cascaded K-Means clustering with ID3 decision tree learning methods. This combined approach showed improved performance measures relative to its individual counterparts. A similar study was done by Yasami et al. [44] where the approach was refined to provide an unsupervised classification for ARP anomaly detection.

Additional variations include *K-Medoids,* which represents clusters with the data median value rather than the mean [42], and *Fuzzy c-means* (i.e. Soft K-Means, Fuzzy K-Means) which is an extension of the K-Means algorithm that allows data points to be associated with multiple clusters using membership values. [42] An improved methodology was proposed by Ensafi et al. [43] which associated Fuzzy K-Means with PSO, denoted as *SFK-means*. This approach provided solutions to the local convergence problem in Fuzzy K-Means and the sharp boundary problem in Swarm K-Means. However, the SFK-means algorithm suffered from a high false positive rate, as well as a high computation overhead in terms of memory requirements and CPU times.

Another methodology offered by Tian et. al. [45], proposed an improved K-Means algorithm that utilized the K-Medoids cyclic method and the improved trilateral relations theorem. This approach produced results that improved the

false detection rate of abnormality and reduced computation time to a certain extent.

2.4 Goals of Next-Generation IDS

Based on the shortcomings of current IDSs, Koch[1] identified a set of unique goals that aid in the development of a Next-Generation IDS. A few of these goals that the proposed methodology focuses on are given below:

1. First, an IDS must aim to support a behavior-based analysis. Due to the increase in the number of Zero Days, targeted attacks, and encrypted communications, it is often not feasible to rely on the availability of signatures. [1] Furthermore, the efficiency of near-real time evaluation of patterns in server systems is limited to both the amount of traffic as well as the size of integrated databases and the quantity of patterns. [1]

2. Second, an IDS must aim to support the exclusion of a learning phase. Many IDSs using Anomaly Detection techniques often require the inclusion of a learning phase; however, retrieving clean labeled data based on a production environment is often unavailable. Thus, Koch[1] recommends the implementation of other techniques that provide solutions for this issue such as unsupervised learning or neural networks.

3. Third, the evaluation of a packet's payload must be abstained, if not prohibited. As the use of encryption increases, the ability to analyze packet payloads becomes very elaborate. Data becomes inaccessible or requires great computational complexity to evaluate. Thus, reliance on the availability of packet payloads becomes increasingly infeasible.[1]

4. Fourth, an IDS must aim to support the implementation of a network-centric design. While host-based implementations hold several advantages with regard to the availability of information (e.g. decrypted data; log files), the management is often complex, error-prone, and limited in scalability. However, network-based installations are able to recognize distributed and sophisticated attacks against the network as a whole. [1]

2.5 Comparison to Works

All of the approaches discussed in the above sections, except Foroushani [49], do not meet all of the goals identified by Koch [1] above for a next-generation intrusion detection system. Foroushani [49] is the only known existing non-intrusive approach that meets all of these goals to date.

This paper will analyze a new non-intrusive approach that is consistent with each of the four goals above required for a next-generation intrusion detection system and compare with Foroushani [49]. [see Fig. 14 pg. 24]

2.6 Contributions

The contributions of the paper are enumerated as follows:

1. The paper presents a non-intrusive approach to detect network intrusions across encrypted accesses using a K-Means clustering algorithm and TCP traffic properties, while maintaining the confidentiality of packet payload information.

2. The paper evaluates the performance of the proposed methodology and compares it with the only other known existing non-intrusive approach using five performance measures. [1,49]

3. The paper presents a non-intrusive approach for analyzing information that is consistent with four of the goals required for a next-generation intrusion detection system.

Chapter 3: Setup and Evaluation

3.1 Setup

For this investigation, a Windows 7 setup will be utilized running a Java 1.7 environment. The specifications of the investigation setup may be found in the table below.

Operating System	*Windows 7 Home Premium SP 1*
System Type	*64-bit*
Memory	*8.00 GB*
Processor	*Intel® Core™ i7-2820QM CPU @ 2.30GHz 2.30GHz*
Hard Drive	*700 GB*

Table 2: System Information

3.2 Data Sets

During the investigation, two evaluation datasets will be used: the KDD Cup '99 dataset and the NSL-KDD dataset.

3.2.1 KDD Cup '99 Dataset

The Knowledge Discovery and Data Mining (KDD) 1999 dataset is derived from the 1998 DARPA Intrusion Detection Evaluation datasets. Under the sponsorship of the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), MIT Lincoln Labs collected and distributed the DARPA datasets for the evaluation of network intrusion detection systems.

The KDD '99 dataset consists of 41 features and one class attribute for each connection record. Of these 41 features, nine are basic features of individual TCP connections. These features may be found in the table below.

Feature Name	Description	Type
Duration	Length (number of seconds of the connection)	Continuous
Protocol_type	Type of the protocol (e.g. tcp, udp, etc.)	Discrete
Service	Network service on the destination (e.g. http, telnet, etc.)	Discrete
Src_bytes	Number of data bytes from source to destination	Continuous
Dst_bytes	Number of data bytes from destination to source	Continuous
Flag	Normal or error status of the connection	Discrete
Land	1 if connection is from/to the same host/port; 0 otherwise	Discrete
Wrong_fragment	Number of "wrong" fragments	Continuous
Urgent	Number of Urgent packets	Continuous

Table 3: Basic features of individual TCP connections [56]

The remaining fields are higher-level features defined by Stolfo et. al. to help in distinguishing normal connections from attacks. [56] These features are grouped in one of several categories.

The first category consists of same-host features. These features examine only connections within the past two second interval that have the same destination host as the current connection and calculate statistics related to protocol behavior, service, and more. [56] Likewise, same-service features examine only connections within the past two second interval that have the same service as the current connection. [56] The same-host and same-service features together are called time-based traffic features of the connection records. [56] These features may be found in the table below.

Feature Name	Description	Type
Count	Number of connections to the same host as the current connection in the past two seconds	Continuous
Note: The following features refer to these same-host connections.		
Serror_rate	% of connections that have "SYN" errors	Continuous
Rerror_rate	% of connections that have "REJ" errors	Continuous
Same_srv_rate	% of connections to the same service	Continuous
Diff_srv_rate	% of connections to different services	Continuous
Srv_count	Number of connections to the same service as the current connection in the past two seconds	Continuous
Note: The following features refer to these same-service connections.		
Srv_serror_rate	% of connections that have "SYN" errors	Continuous
Srv_rerror_rate	% of connections that have "REJ" errors	Continuous
Srv_diff_host_rate	% of connections that have different hosts	Continuous

Table 4: Time-based traffic features [56]

The second category contains host-based traffic features. [56] These features were derived with respect to attacks that use a much larger time interval than two seconds (e.g. probing attacks scanning once per minute). As such connection records were sorted by destination host, and features were constructed using a

window of 100 connections to the same host instead of a time window. [56] These features may be found in the table below.

Feature Name	Description	Type
Dst_host_count	Number of connections to the same host as the current connection.	Continuous
Dst_host_srerror_rate	% of connections that have "SYN" errors	Continuous
Dst_host_rerror_rate	% of connections that have "REJ" errors	Continuous
Dst_host_same_srv_rate	% of connections to the same service	Continuous
Dst_host_diff_srv_rate	% of connections to different services	Continuous
Dst_host_srv_count	Number of connections to the same service as the current connection.	Continuous
Dst_host_srv_serror_rate	% of connections to same service that have "SYN" errors	Continuous
Dst_host_srv_rerror_rate	% of connections to same service that have "REJ" errors	Continuous
Dst_host_srv_diff_host_rate	% of connections to different hosts	Continuous
Dst_host_same_src_port_rate	% of connections that were to the same source port	Continuous

Table 5: Host-based traffic features [71]

The last category consists of content features. [56] Unlike DoS and Probing attacks, R2L and U2R attacks do not have intrusion frequent sequential patterns. [54] This is mainly due in part to DoS and Probing attacks utilizing many connections to some host(s) in a short period of time, while R2L and U2R attacks are embedded in the data portions of the packets utilizing only a single connection. [54] Thus, in order to detect these kinds of attacks, content features were added using domain knowledge, in order to look for suspicious behavior in the data portions (e.g. number of failed login attempts). [56] A list of these features may be found in the table below.

Feature Name	Description	Type
Hot	Number of "hot" indicators	Continuous
Num_failed_logins	Number of failed login attempts	Continuous
Logged_in	1 if successfully logged in; 0 otherwise	Discrete
Num_compromised	Number of "compromised" conditions	Continuous
Root_shell	1 if root shell is obtained; 0 otherwise	Discrete
Su_attempted	1 if "su root" command attempted; 0 otherwise	Discrete
Num_root	Number of "root" accesses	Continuous
Num_file_creations	Number of file creation operations	Continuous
Num_shells	Number of shell prompts	Continuous
Num_access_files	Number of operations on access control files	Continuous
Num_outbound_cmds	Number of outbound commands in an ftp session	Continuous
Is_hot_login	1 if the login belongs to the "hot" list; 0 otherwise	Discrete
Is_guest_login	1 if the login is a "guest" login; 0 otherwise	Discrete

Table 6: Content features within a connection suggested by domain knowledge [56]

Lastly, the class attribute consists of 1 of 21 classes that fall under four types of attacks [55-56]. These classes and attack types may be found in the tables below.

Class Name	Attack Type
Back	DoS
Buffer_overflow	U2R
ftp_write	R2L
Guess_passwd	R2L
Imap	R2L
Ipsweep	Probe
Land	DoS
Loadmodule	U2R
Multihop	R2L
Neptune	DoS
Nmap	Probe
Perl	U2R
Phf	R2L
Pod	DoS
Portsweep	Probe
Rootkit	U2R
Satan	Probe
Smurf	DoS
Spy	R2L
Teardrop	DoS
Warezclient	R2L
Warezmaster	R2L

Table 7: List of Classes [56]

Attack Type	Description
Probing (Probe)	Surveillance and other probing (e.g. port scanning)
Remote-to-Local (R2L)	Unauthorized access from a remote machine (e.g. guessing password)
User-to-Root (U2R)	Unauthorized access to local superuser (root) privileges (e.g. buffer overflow attacks)
Denial-of-Service (DoS)	Denial of Service (e.g. syn flood)

Table 8: Types of attacks [56]

Although the KDD Cup '99 dataset has become a widely used dataset for the evaluation of detection systems, it does suffer from a few shortcomings [54]. As such, a newer dataset was proposed to handle many of these issues, called the NSL-KDD dataset. [54]

3.2.2 NSL-KDD Dataset

The NSL-KDD dataset is a reduced version of the original KDD '99 dataset to handle many of the KDD '99 shortcomings. [55] It was proposed by Tavallaee et al. [54] in 2009 and includes some of the following differences over the original KDD '99 dataset:

1. First, the training set does not include redundant records. This aids in preventing the classifiers from being biased towards frequent records. [57]
2. Second, the test sets do not include duplicate records. This prevents the performance of learning algorithms from being biased towards methods which yield better detection rates on frequent records. [57]
3. Third, the number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD '99 dataset. As a result, the classification rates of distinct learning methods vary in a wider range, increasing the accuracy of evaluation of different learning techniques. [57]
4. Last, the number of records in both the training and test datasets are reasonable, allowing experiments to run the complete datasets without the need to randomly select a smaller portion. As such, this allows the evaluation results of different research works to be more consistent and comparable. [57]

3.3 Proposed System

The proposed intrusion detection system (IDS) will consist of three primary areas. [15]

1. Data Initialization
2. K-Means Clustering (Unsupervised)
3. Intrusion Detection Analysis

An overview of this process may be found in the figure below.

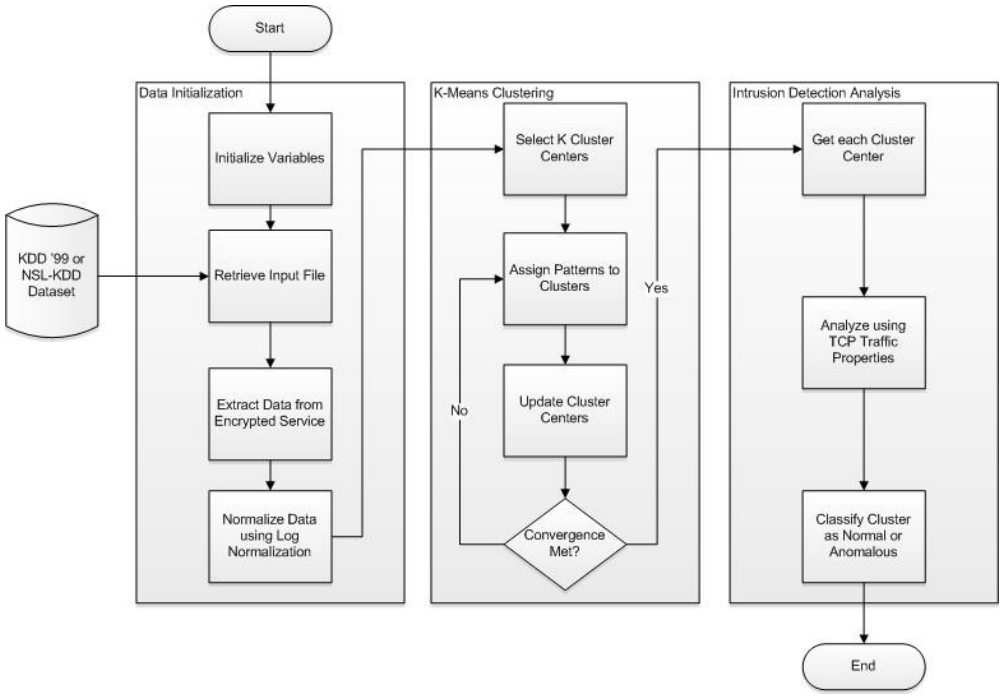

Figure 14: Overview of System Specifications

First, the system will extract information from a respective dataset. This information includes intrinsic attributes from the header's area of network packets (i.e. TCP traffic properties), time-based attributes, host-based attributes, content attributes, and a class attribute each related to the network data. [71] A list of each of these attributes with their respective descriptions may be found above in the dataset section in Table 3 through Table 8.

Then, once the information is retrieved, the K-Means Clustering algorithm will sort the data into cluster groups for analysis. This will allow for various actions to be identified based on the similarity of the information. In order to accomplish this task, four primary steps will be utilized: selecting *K* cluster centers to initialize each cluster group, assigning data patterns to cluster groups using the cluster centers, updating the cluster centers using the new pattern sets, and then checking for convergence of the algorithm based on a stopping criterion.

Finally, attacks will be detected according to the TCP traffic properties of the cluster centers. This will include features such as the number of bytes from source to destination and vice-versa.

The next set of subsections will discuss the applied methodology in more detail.

3.3.1 Data Initialization

The first area of the proposed IDS is the data initialization module. In this phase, traffic data is extracted and formatted to be analyzed for the K-Means clustering algorithm. An overview of this process may be found in the figure below.

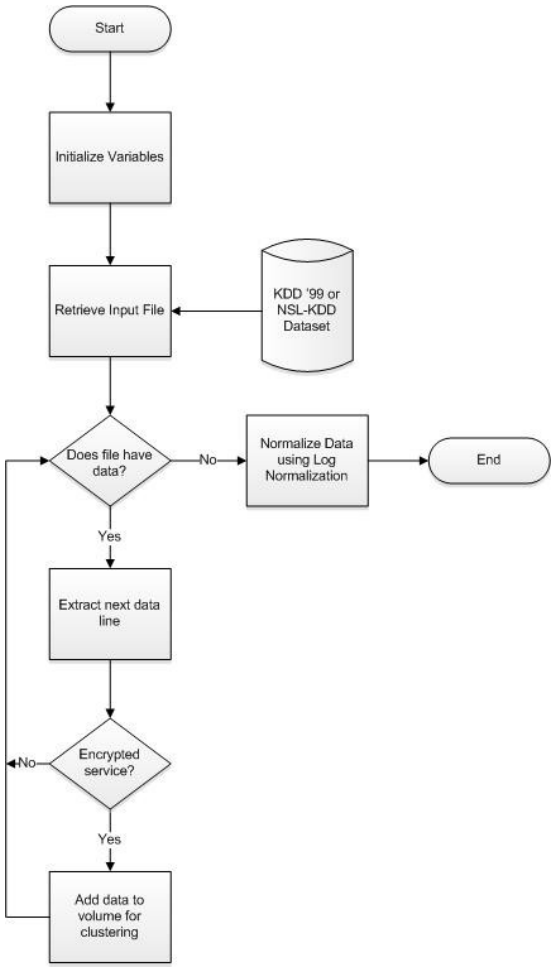

Figure 15: Overview of initializing variables and retrieving data for clustering

In the first step, system variables are initialized for processing. This includes essential information such as the name and location of the dataset to be processed (e.g. KDD or NSL-KDD), the token for how the data is delimited, the volume to store the data patterns as they are retrieved, as well as additional variables that will be used throughout the experiment.

Next, information from the selected dataset is retrieved through the use of a filereader. The "kddcup.data" file (KDD) and the "KDDTrain+" file (NSL-KDD) will be used to evaluate the proposed system, since they each contain the largest quantity of labeled evaluation data from their respective datasets. This will allow the data to be compared to a known standard of effectiveness in order to validate

the methodology, once the analysis phase has completed. Additionally, as each data element is retrieved, the service feature of the element is analyzed for protocols supporting encrypted communications during the collection of the original datasets. This will allow the system to analyze data features that supported encrypted communications, restricting use of the packet information. For this experiment, Secure Shell (i.e. SSH) traffic data will be focused on and extracted from each of the datasets. The details and information related to each of the specific dataset files may be found in the table below.

	KDD Cup '99	NSL-KDD
Total Data Lines	4,898,431	125,973
Total Passive Activities	972,780	67,343
Total Intrusive Activities	3,925,651	58,630
Total DoS Attacks	3,883,370	45,927
Total U2R Attacks	52	52
Total R2L Attacks	1,126	995
Total Probe Attacks	41,102	11,656
Total SSH Data Lines	1,075	311
Total SSH Passive Activities	7	5
Total SSH Intrusive Activities	1,068	306
Total SSH DoS Attacks	1,039	281
Total SSH U2R Attacks	0	0
Total SSH R2L Attacks	0	0
Total SSH Probe Attacks	29	25

Table 9: KDD and NSL-KDD Dataset Information

As each data element is extracted, it is added to a preformatted array list, or volume. Once this operation has completed, each set of data is normalized using a log transform strategy. The equation for the log normalization may be found area below.

$$X'_{ij} = \log(1 + X_{ij})$$

Where X_{ij} = the value of feature j for observation i before normalization

X'_{ij} = the value after normalization is applied

By normalizing the data, it allows for the contributions of different data attributes to be weighed in a manner such that the distance between observations becomes meaningful. [75] This will allow the K-Means clustering algorithm to begin to process and group the extracted data.

3.3.2 K-Means Clustering (Unsupervised)

In the second area of the proposed IDS, the K-Means clustering module sorts the data into cluster groups for analysis. The algorithm accomplishes this task through four steps, as seen in the figure below.

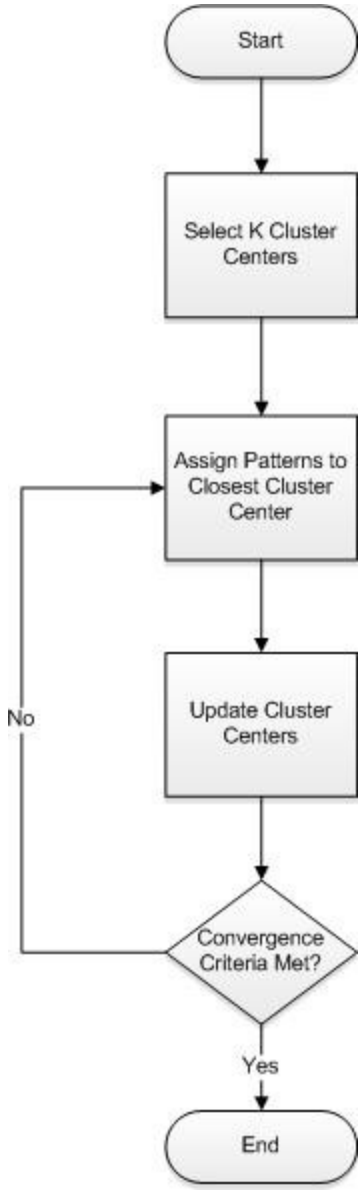

Figure 16: Overview of K-Means Clustering Algorithm

First, *k* cluster centers are chosen to coincide with *k* randomly defined points from the respective dataset. In order to accomplish this task, a strategy similar to the Forgy methodology is implemented, where *k* pattern sets (i.e. data lines) are randomly selected from the volume containing the selected dataset and used as the initial cluster means, or centers. [58]

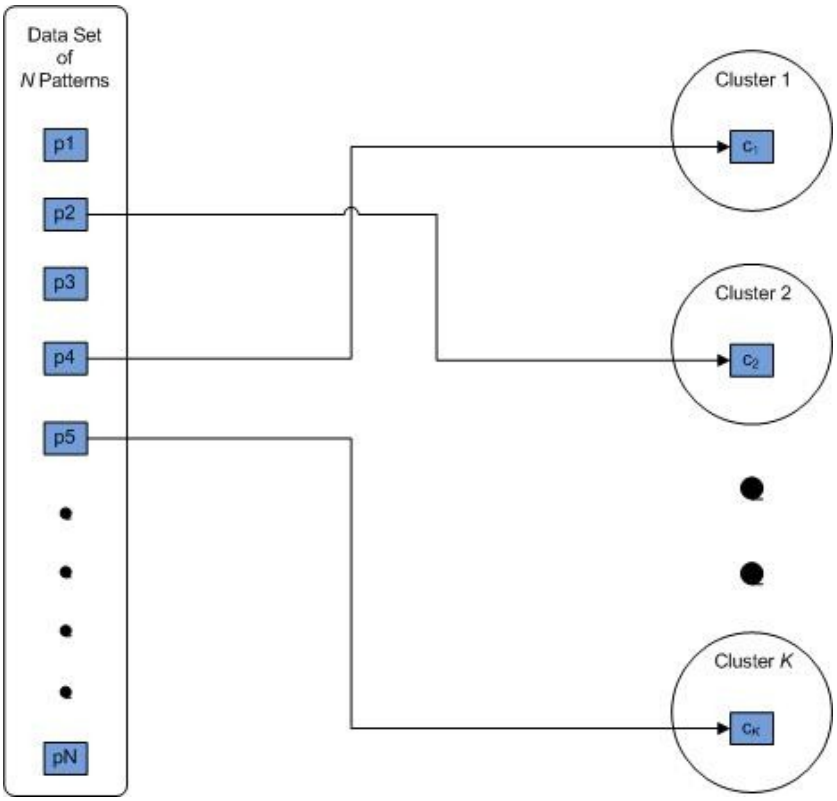

Figure 17: Selection of K Cluster Centers

For this experiment, the initial starting value of *K* will be set to the general rule of thumb proposed by Mardia et al. [59]:

$$K = \sqrt{\frac{n}{2}}$$

Where n = total number of data patterns

As such, using the information provided in Table 9 above, the *K* values for each dataset can be calculated. These values may be found in the table below.

	Total Data Lines	*K (Total)*	Total SSH Data Lines	*K (SSH)*
KDD Cup '99	4,898,431	1,564	1,075	23
NSL-KDD	125,973	250	311	12

Table 10: KDD and NSL-KDD Dataset K Values

The KDD Cup '99 dataset contains 4,898,431 total patterns with 1,075 SSH specific data patterns. This generates a *K* value of 1,564 clusters when operating on the complete data set, and a *K* value of 23 when examining the SSH specific information. Likewise, the NSL-KDD dataset contains 125,973 total patterns with 311 SSH specific data patterns. As such, this generates a *K* value of 250 clusters when operating on the full data set, and a *K* value of 12 when examining the SSH information.

Each cluster center, or centroid (*c*), is also associated with the mean value of that cluster's assigned pattern set. Thus, as patterns are assigned to clusters, the cluster centroids are able to be updated as needed.

Next, each pattern from the respective dataset is assigned to a cluster. This is accomplished by comparing each pattern (*p*) to each cluster centroid (*c*) through the implementation of the Euclidean Distance function. An overview of the process may be found in the figure below.

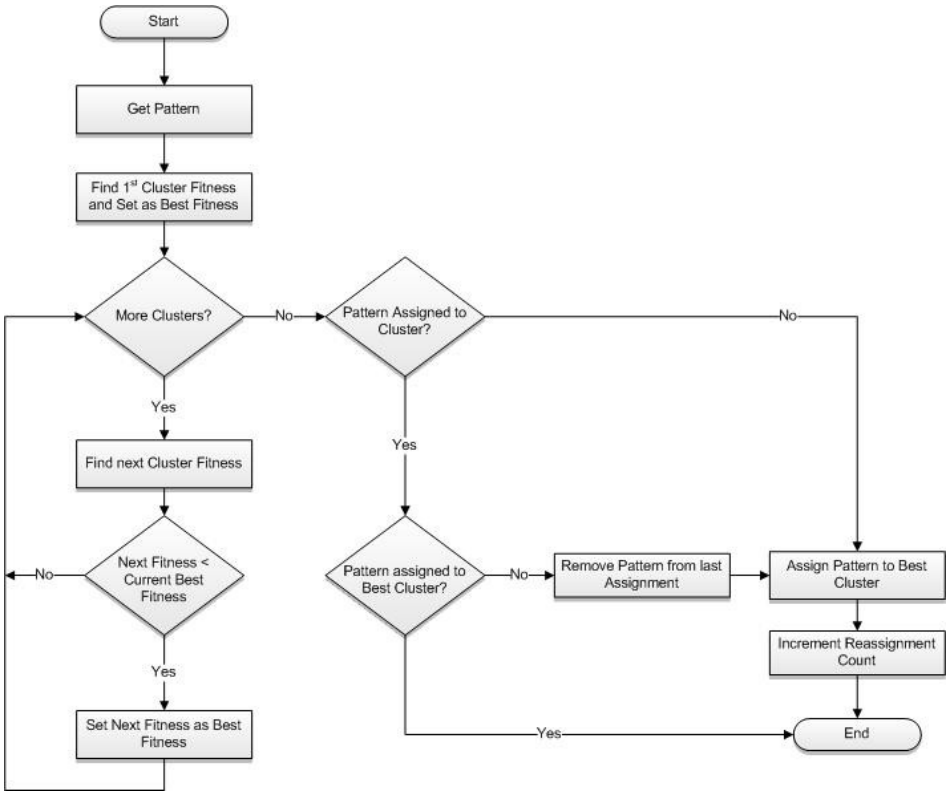

Figure 18: Pattern Assignment

As each pattern is examined, a fitness value is generated for each cluster, using the distance function. This fitness value relates the distance of a single data pattern to a cluster's centroid. As such, the cluster with the lowest fitness value is the one whose centroid produced the least Euclidean Distance to the pattern.

In order to accomplish this task, each pattern (p) from the data volume is examined and compared with each respective cluster centroid (c), such that the Euclidean Distance is given as:

$$d(p,c) = \sqrt{\sum_{i=1}^{N}(p_i - c_i)^2}$$

Where n = total number of elements in a data pattern

p_i= the ith pattern element

c_i= the ith centroid element

$d(p,c)$= distance from pattern p to centroid c

For this experiment, the fitness values generated for each pattern will focus on the continuous intrinsic attributes from the header's area of the network packets (i.e. TCP traffic properties). These attributes may be found in the table below.

Feature Name	Description
Duration	Length (number of seconds of the connection)
Src_bytes	Number of data bytes from source to destination
Dst_bytes	Number of data bytes from destination to source
Wrong_fragment	Number of "wrong" fragments
Urgent	Number of Urgent packets

Table 11: Continuous intrinsic attributes of KDD and NSL-KDD datasets [56]

By utilizing this information, it will allow the algorithm to focus on core traffic features that are most relevant in detecting attacks [73], as well as features that are usually available in network packets during encrypted communications. Furthermore, it will mitigate the reliance on higher-level features that are less relevant in detecting attacks [73] and that would not usually be available without additional data analysis.

Once each fitness value is obtained, a pattern can be assigned to its best matching cluster, or the cluster that produced the lowest distance to the pattern. If the pattern was previously assigned to a cluster, then this assignment can also be updated with each iteration of the algorithm. Lastly, a reassignment counter is also incremented as each pattern is assigned or reassigned. This will aid in determining if the clustering has reached a convergence criteria.

After each pattern has been assigned, each cluster centroid is recomputed as the new average, or mean, of the respective cluster's assigned patterns, or pattern set. In order to accomplish this task, each cluster's pattern set is traversed, retrieving each pattern element and summing the value together with the complementary centroid element. Once this process is complete, each centroid element is divided by the total number of assigned patterns plus one for the centroid, in order to find the mean pattern set for that cluster. An overview of this process may be found in the figure below.

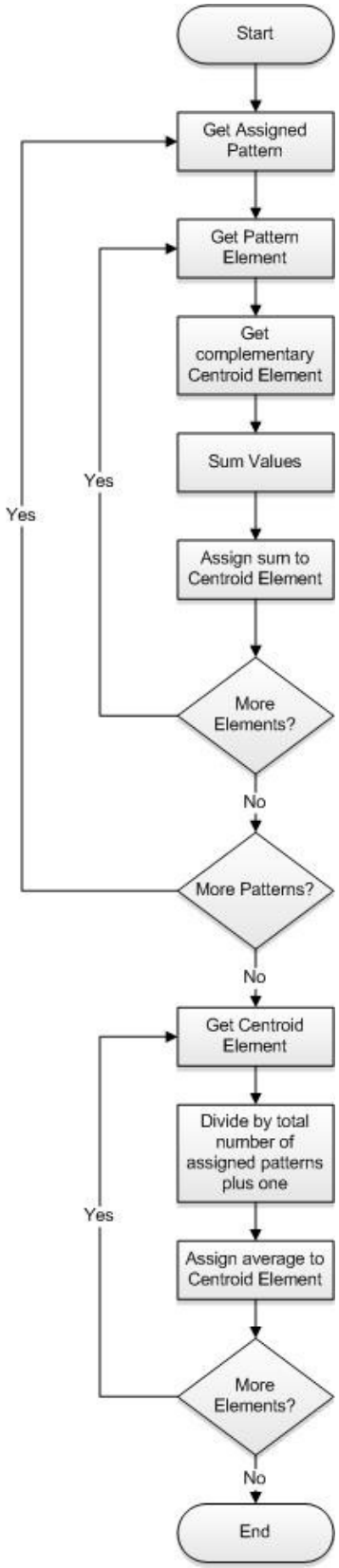

Figure 19: Updating a Cluster Centroid

Finally, the convergence criteria is checked to see if the algorithm has reached its stopping point. If this criteria has been met, then the K-Means clustering has completed sorting the data and the intrusion detection module may begin its analysis. Otherwise, the second and third steps of the K-Means algorithm are repeated until the convergence criteria has been met.

For this experiment, the convergence criteria will be set to be a minimum number of pattern reassignments across each clustering iteration. That value will be 1/8 of the total number of patterns.

Finally, Java 1.7 multithreading will also be implemented across pattern assignment and centroid updates, in order to fully utilize available setup resources.

3.3.3 Intrusion Detection Analysis

The third area of the proposed IDS is the intrusion detection module. In this phase, the final clustering of the dataset is analyzed to determine if each set of cluster patterns is normal or potentially intrusive. As such, a non-intrusive approach is taken that is similar to Foroushani et al. [49], where each cluster centroid is analyzed for abnormal activity based on the TCP traffic properties of the related network data (i.e. intrinsic attributes). Foroushani et al. [49] also discussed how SSH packet sizes may be computed based on the protocol's specifications. From this information, the traffic properties for incoming and outgoing data bytes can be used to analyze each cluster group without relying on detailed payload information. An overview of this process may be found in the figure below.

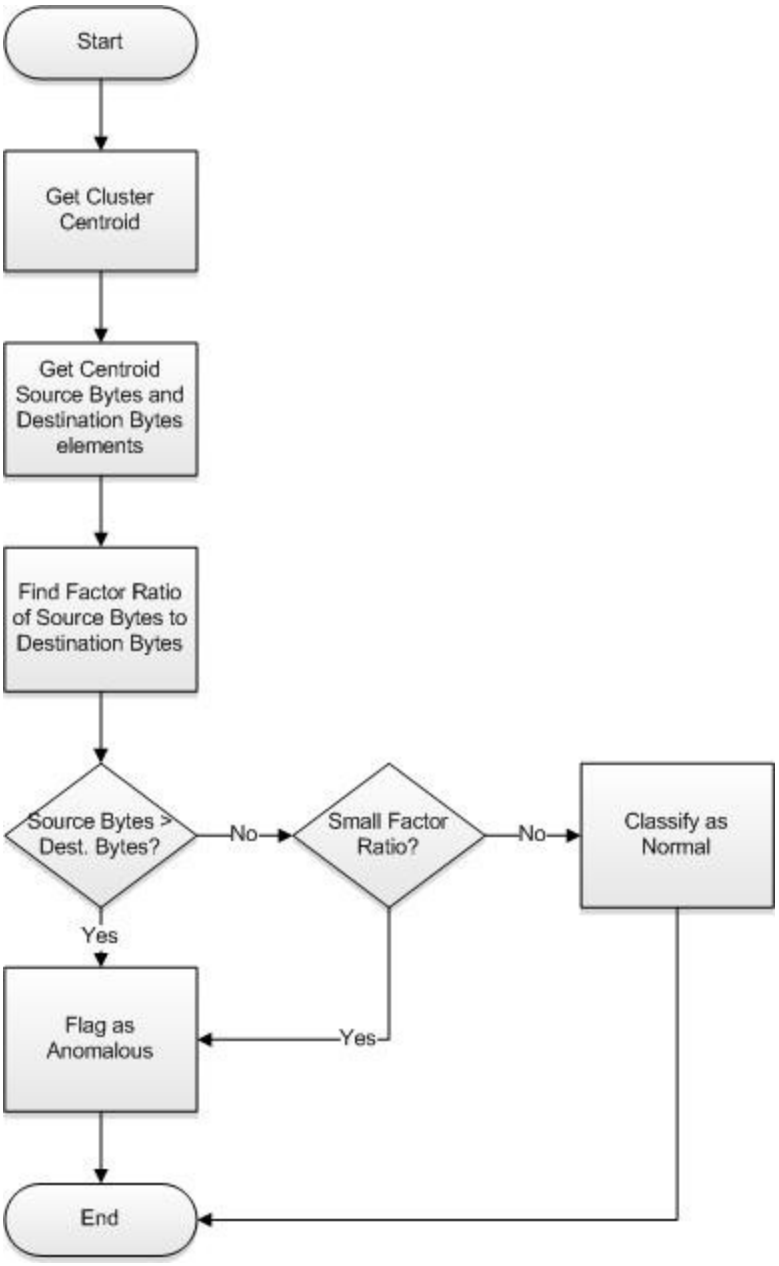

Figure 20: Intrusion Detection Analysis using TCP Traffic Properties

As each cluster centroid is examined, the traffic properties for source and destination bytes are retrieved. The descriptions for each of these properties may be found in the table below. Additional information may also be found in Table 3 of the data set section.

Feature Name	Description
src_bytes	Number of data bytes from source to destination
dst_bytes	Number of data bytes from destination to source

Table 12: Data byte features of individual TCP connections [56]

Using this information, the TCP traffic properties can be examined based on the expected network behavior. For example, in normal SSH access, the request, or input traffic size, is usually small while the reply, or output traffic size, is large. [49] Thus, a request with a large input size and a reply with a small output size can possibly be an intrusion. [49] Likewise, in typical scanning or probe attacks, the replies are smaller than regular contents, even if the requests are similar to normal traffic. [49] This is because that a server that is not vulnerable typically sends a response that includes a small message with an error status. [49]

As such, each cluster centroid's traffic properties are examined for potential intrusions using this criterion. [49] For this experiment, a factor ratio will be computed in order to relate the value of source bytes to the value of destination bytes. If the source bytes feature is greater than the destination bytes property (e.g. buffer overflow attacks), the cluster is flagged as potentially intrusive. Likewise, if the factor ratio is small (e.g. scanning attacks), a flag will also be set. However, if the centroid passes each of the traffic property evaluations, then it is classified as normal. For this experiment, the minimum factorial ratio value was set to ten.

The java implementation of the proposed system may be found in Appendix B.

3.4 Evaluation

During this investigation, the precision and recall of the proposed system will be collected and analyzed with each dataset implementation, in order to evaluate the system's effectiveness in detecting intrusions. [60] The precision is denoted as the fraction of retrieved instances that are relevant, or the fraction of potentially intrusive activities that are flagged with respect to the total number of alerts that are generated. Likewise, the recall is the fraction of relevant instances that are retrieved, or the fraction of potentially intrusive activities that are flagged with respect to the total number of intrusive activities that are present.

The false rate of the system will also be analyzed in order to evaluate the efficiency of the non-intrusive approach against the only other known existing non-intrusive methodology. [1,49] The false rate is denoted as the summation of both the false positive rate and the false negative rate. [49] As such, it is composed of the total number of false alarms with respect to the total number of passive activities, as well as the total number of false negatives with respect to the total number of intrusive activities. An overview of each of the measurements may be found in the table below.

Measurement Name	Description
Precision	Fraction of flagged intrusive activities w.r.t. the total number of generated alerts.
Recall	Fraction of flagged intrusive activities w.r.t. the total number of intrusive activities present.
False Positive Rate (FPR)	Fraction of false alarms generated (e.g. flagged passive activities) w.r.t. the total number of passive activities.
False Negative Rate (FNR)	Fraction of false negatives generated (e.g. intrusive activities that were not flagged) w.r.t. the total number of intrusive activities.
False Rate	The summation of both the false positive rate and the false negative rate.

Table 13: Measurement Descriptions

The precision, recall, and false rate values will be collected for 25 trials on each dataset, along with relative time values. Once the trials have been completed, the average precision, recall, and false rate values will be calculated. The equations for each of these measurements may be found below.

$$Recall = \frac{D}{B + D}$$

$$Precision = \frac{D}{C + D}$$

$$False\ Positive\ Rate\ (FPR) = \frac{C}{A + C}$$

$$False\ Negative\ Rate\ (FNR) = \frac{A}{A + D}$$

$$False\ Rate = FPR + FNR$$

Where A = True Negatives

B = False Negatives

C = False Positives

D = True Positives

Chapter 4: Results and Conclusions

This chapter is divided into three sections. The first section displays the results from the evaluation of the proposed system. This includes measurement values and discussions as to what can be inferred from the retrieved data. The second and third sections discuss the final conclusions that can be drawn from the results, the limitations with the system, and the recommendations for future work.

4.1 Results

4.1.1 Precision

The following table reflects the precision data collected for 25 trials on each dataset.

Dataset	Average Precision	Standard Deviation
KDD	100.00%	±0.00%
NSL-KDD	100.00%	±0.00%

Table 14: Average Precision Values for KDD and NSL-KDD Datasets

By examining this information, it shows that the fraction of retrieved instances that are relevant remained at a high value across each trial evaluation. As such, passive activities were able to be classified appropriately across each trial, minimizing false positives from the data.

4.1.2 Recall

The following table reflects the recall data collected for 25 trials on each dataset.

Dataset	Average Recall	Standard Deviation
KDD	99.06%	±0.60%
NSL-KDD	97.86%	±1.79%

Table 15: Average Recall Values for KDD and NSL-KDD Datasets

By examining this information, it shows that the fraction of relevant instances that were retrieved remained at a high value across most of the trials. However, while many intrusive activities were able to be classified correctly as anomalies, there were still a fair amount of false negatives, or intrusive activities that were not flagged.

4.1.3 False Rate

The following table reflects the false rate data collected for 25 trials on each dataset.

Dataset	Average False Rate	Standard Deviation
KDD	0.94%	±0.601%
NSL-KDD	2.14%	±1.788 %

Table 16: Average False Rate Values for KDD and NSL-KDD Datasets

By examining this information, it shows that both the false positive rate and the false negative rate were able to remain within a small value range across each set of trials. As such, many of the passive activities and intrusive activities were able to be classified appropriately, containing only a small quantity of passive activities that were flagged or intrusive activities that were not flagged.

4.1.4 Time

The following table reflects the time data collected for 25 trials on each dataset.

Dataset	Average Time (in seconds)	Standard Deviation (in milliseconds)	Average # of Loops
KDD	00:27:618	±00:00:189	3
NSL-KDD	00:01:191	±00:00:017	3

Table 17: Average Time Values for KDD and NSL-KDD Datasets

By examining this information, it shows that average time required to perform each trial was within consistent time frames with respect to the average number of iterations used by the K-Means clustering algorithm.

4.1.5 Dataset Comparison

The following table shows the relationship of the number of records to processing time for each dataset implementation.

Dataset	Total Records	SSH Records	Average Time
KDD	4,898,431	1,075	00:27:618 seconds
NSL-KDD	125,973	311	00:01:191 seconds

Table 18: Number of Records and Processing Time

The average results for each dataset are summarized in the table below. The confusion matrix for each dataset is also located below the summary table.

	KDD		NSL-KDD	
	Average	*Standard Deviation*	*Average*	*Standard Deviation*
Avg Precision	100.00%	±0.00%	100.00%	±0.00%
Avg Recall	99.06%	±0.60%	97.86%	±1.79%
Avg False Rate	0.94%	±0.601%	2.14%	±1.788 %
Avg Time	00:27:618	±00:00:189	00:01:191	±00:00:017

Table 19: Dataset Comparison Summary

KDD	Flagged	Not Flagged
Intrusive	1054	14
Passive	0	7

Table 20: Confusion Matrix for KDD

NSL-KDD	Flagged	Not Flagged
Intrusive	295	11
Passive	0	5

Table 21: Confusion Matrix for NSL-KDD

4.1.6 Non-Intrusive Analysis Comparison

The average false rate results from each non-intrusive methodology are summarized in the table below.

	Proposed System		Foroushani[49]
	KDD	NSL	
Avg False Rate	0.94%	2.14%	15%

Table 22: Non-Intrusive Analysis Comparison Summary

By examining this information, it shows that the average false rate for the proposed non-intrusive approach performed well in comparison to the only other known existing non-intrusive methodology.

4.2 Conclusions

In this paper, a non-intrusive approach was proposed to detect network intrusions across encrypted accesses using a K-Means clustering model and TCP traffic properties. While many Intrusion Detection Systems will typically utilize a signature-based or anomaly-based approach to analyze in-the-clear network traffic, the growing use of encrypted communications continues to deny the use of payload-related data. This non-intrusive methodology provides a potential solution to mitigate these issues, by complying with four goals required for next-generation intrusion detection systems. [1]

This approach supports a behavior-based analysis, in contrast to the reliance on databases of attack signatures. Thus, this approach should provide support for the growing use of encrypted communications. An unsupervised K-Means clustering algorithm was used so that there is no learning phase. There was thus no evaluation of packet payload-related data. This also ensures the confidentiality of the packet information and reduces the computational complexity of the evaluation. This work used a non-intrusive approach similar to Foroushani et. al. [49]. This uses TCP traffic properties for evaluation of the data. Lastly, this approach supports the implementation of a network-centric design. There was thus no reliance on host-based implementation data for evaluation (e.g. log files). The KDD '99 and NSL-KDD evaluation datasets were used to test this new approach.

This paper analyzes a new non-intrusive approach that is able to detect many intrusions in the used datasets. Furthermore, the average false detection rate of the tested system showed greatly improved results in comparison to other non-intrusive methodologies. Therefore, based on the tables in chapter 4, the proposed system operated effectively in detecting many of the network intrusions for these datasets while maintaining the encrypted confidentiality of the packet information.

4.3 Future Work

Now that the investigation has concluded, there are some extensions of this work.

First, implement an enhanced clustering algorithm that shortens the time to group the data. While the K-Means clustering provides a methodology to sort information into similar cluster groups, it has inherent drawbacks, including the ability to fall into a local optimum instead of a global optimum as well as potentially creating an exponential running-time as the dataset information continues to scale in size. [74] With this route, there are a few interesting areas of study analyzing clustering algorithms with real-time data. [61-63]

Second, a different standardized dataset of traffic information should be tested. This should include a more balanced distribution of data between both passive and intrusive activities and a more diverse set of attacks. The KDD and NSL-KDD datasets are useful in analyzing the system against basic network-related data. However, each of the datasets used herein suffer from limitations. These include a strong correlation towards intrusive network behaviors and limited types of attack types for encrypted service protocols (e.g. SSH). Examples of these features may be seen in Table 9 above.

Third, the use of additional traffic properties can also be examined. [49]

Chapter 5: Bibliography

[1] Koch, R., "Towards next-generation Intrusion Detection," *Cyber Conflict (ICCC), 2011 3rd International Conference on* , vol., no., pp.1-18, 7-10 June 2011
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5954707&isnumber=5954687

[2] "Intrusion Detection Systems," Information Assurance Technology Analysis Center (IATAC), Herndon, VA, OMB No. 0704-0188, September 25, 2009.

[3] Thomas Whitehead. (2013, July 5). *Club Nintendo Japan Falls Victim to Hack Attack* [Online]. Available: http://www.nintendolife.com/news/2013/07/club_nintendo_japan_falls_victim_to_hack_attack

[4] Chris Davies. (2013, July 5). *Club Nintendo Japan Hacked* [Online]. Available: http://www.slashgear.com/club-nintendo-japan-hacked-05289196/

[5] Andy Green. (2013, July 3). *Ubisoft Hit By Hacking Attack* [Online]. Available: http://www.nintendolife.com/news/2013/07/ubisoft_hit_by_hacking_attack

[6] Patrick Seybold. (2011, April 22). *Update on PlayStation Network/Qriocity Services* [Online]. Available: http://blog.us.playstation.com/2011/04/22/update-on-playstation-network-qriocity-services/

[7] (2011, April 28). *Sony faces legal action over attack on PlayStation network* [Online]. Available: http://www.bbc.co.uk/news/technology-13192359

[8] (2011, May 17). *PlayStation Network Restoration Begins* [Online]. Available: http://uk.playstation.com/psn/news/articles/detail/item369506/PSN-Qriocity-Service-Update/

[9] (2011, June 3). *Sony investigating another hack* [Online]. Available: http://www.bbc.co.uk/news/business-13636704

[10] Ghosh, Anup et al., "A Real-Time Intrusion Detection System Based on Learning Program Behavior," *Recent Advances in Intrusion Detection, in Lecture Notes in Computer Science*, LNCS 1907, Springer Berlin / Heidelberg, 2000

[11] Deepak Gautam. (2013, June 2). *Host Based Intrusion Detection System (HIDS)* [Online]. Available: http://deepakgautam.com.np/2013/06/host-based-intrusion-detection-system-hids/

[12] Sailesh Kumar. (2007, December). *Survey of Current Network Intrusion Detection Techniques* [Online]. Available: http://www.cse.wustl.edu/~jain/cse571-07/ftp/ids/

[13] Denning, D.E., "An Intrusion-Detection Model," in *Software Engineering, IEEE Transactions on* , vol.SE-13, no.2, pp.222,232, Feb. 1987, doi: 10.1109/TSE.1987.232894,

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1702202&isnumber=35884

[14] Sabahi, F.; Movaghar, A., "Intrusion Detection: A Survey," *Systems and Networks Communications, 2008. ICSNC '08. 3rd International Conference on* , vol., no., pp.23,26, 26-31 Oct. 2008, doi: 10.1109/ICSNC.2008.44,
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4693640&isnumber=4693626

[15] D. J. Brown et al. (2002). *A Survey of Intrusion Detection Systems* [Online]. Available: http://charlotte.ucsd.edu/classes/fa01/cse221/projects/group10.pdf

[16] Guy Bruneau, "The History and Evolution of Intrusion Detection," SANS Inst., Bethesda, MD, ver. 1.2f, 2001.

[17] James P. Anderson. (1972, October). *Computer Security Technology Planning Study Volume 2* [Online].
Available: http://seclab.cs.ucdavis.edu/projects/history/papers/ande72.pdf

[18] James P. Anderson. (1980, April 15). *Computer Security Threat Monitoring and Surveillance* [Online].
Available: http://seclab.cs.ucdavis.edu/projects/history/papers/ande80.pdf

[19] Denning, D.E., "An Intrusion-Detection Model," *Software Engineering, IEEE Transactions on* , vol.SE-13, no.2, pp.222,232, Feb. 1987 doi: 10.1109/TSE.1987.232894, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1702202&isnumber=35884

[20] *Next-Generation Intrusion-Detection Expert System (NIDES)* [Online]. Available: www.csl.sri.com/projects/nides/

[21] *Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD)* [Online]. Available: www.csl.sri.com/projects/emerald/

[22] Kemmerer, R.A.; Vigna, G., "Intrusion detection: a brief history and overview," *Computer* , vol.35, no.4, pp.27,30, Apr 2002 doi: 10.1109/MC.2002.1012428, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1012428&isnumber=21810

[23] (2009). *Finjan Secure Web Gateway Solutions for Enterprises* [Online]. Available: http://www.virusdefence.com.au/finjan/finjan-zero-hour.html

[24] Yan, K.Q.; Wang, S.C.; Wang, S.S.; Liu, C. W., "Hybrid Intrusion Detection System for enhancing the security of a cluster-based Wireless Sensor Network," *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on* ,

vol.1, no., pp.114,118, 9-11 July 2010
doi: 10.1109/ICCSIT.2010.5563886,
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5563886&isnumber=5563682

[25] Yan, K.Q.; Wang, S.C.; Wang, S.S.; Liu, C. W., "An Integrated Intrusion Detection System for Cluster-based Wireless Sensor Networks," in *Expert Systems with Applications* , vol.38, no. 12, pp. 15234-15243, November-December 2011. doi: http://dx.doi.org/10.1016/j.eswa.2011.05.076,
URL: http://www.sciencedirect.com/science/article/pii/S0957417411008608

[26] (2008). *How Does Anti virus detects viruses?* [Online]. Available: http://www.breakthesecurity.com/2011/05/how-does-anti-virus-detects-viruses.html

[27] *R Interface to Oracle Data Mining* [Online]. Available: http://www.oracle.com/technetwork/database/options/advanced-analytics/odm/odm-r-integration-089013.html

[28] (2013). *Corporate Overview* [Online]. Available: https://www.damballa.com/company/

[29] Yingbing Yu, "A survey of anomaly intrusion detection techniques," *J. Comput. Sci. Coll.,* vol. 28, no. 1, pp. 9-17, October, 2012.

[30] Varun Chandola et al., “Anomaly detection: A Survey,” *ACM Comput. Surv.,* vol. 41, no. 15, July, 2009.

[31] Donald Tabone. (2007, April 5). *The concept of Intrusion Detection Systems* [Online]. Available: http://maltainfosec.org/archives/26-The-concept-of-Intrusion-Detection-Systems.html

[32] “Security Labs Report,” M86 Security, 2011

[33] “Internet Security Threat Report,” Symantec, (Mountain View, CA), April 2013, vol. 18

[34] Ms. Smith. (2013, April 23). *Verizon report: China behind 96% of all cyber-espionage data breaches* [Online]. Available: http://www.networkworld.com/community/blog/verizon-report-china-behind-96-all-cyber-espionage-data-breaches

[35] Estivill-Castro, V., "Why so many clustering algorithms - A Position Paper," *ACM SIGKDD Explorations Newsletter*, vol.4, no. 1, pp.65-75, June 2002 doi: 10.1145/568574.568575

[36] Jain, A.K.; Murty, M.N.; Flynn, P.J., "Data Clustering: A Review," *ACM Computing Surveys (CSUR)*, vol.31, no. 3, pp.264-323, September 1999 doi: 10.1145/331499.331504

[37] Dr. Hans-Peter Kriegel. *Density-Based Cluster- and Outlier Analysis* [Online]. Available: http://www.dbs.informatik.uni-muenchen.de/Forschung/KDD/Clustering/

[38] Lizhong Xiao; Zhiqing Shao; Gang Liu, "K-means Algorithm Based on Particle Swarm Optimization Algorithm for Anomaly Intrusion Detection," *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on* , vol.2, no., pp.5854,5858, 0-0 0 doi: 10.1109/WCICA.2006.1714200, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1714200&isnumber=36092

[39] Miranda Irene. (1999, April 1). *Square error clustering methods* [Online]. Available: http://www.cse.iitb.ac.in/dbms/Data/Courses/CS632/1999/clustering/node17.html

[40] Dr. Saed Sayad. (1999, April 1). *K-Means Clustering* [Online]. Available: http://www.saedsayad.com/clustering_kmeans.htm

[41] Gaddam, S.R.; Phoha, V.V.; Balagani, K.S., "K-Means+ID3: A Novel Method for Supervised Anomaly Detection by Cascading K-Means Clustering and ID3 Decision Tree Learning Methods," *Knowledge and Data Engineering, IEEE Transactions on* , vol.19, no.3, pp.345,354, March 2007 doi: 10.1109/TKDE.2007.44, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4072746&isnumber=4072743

[42] Jain, A.K., "Data Clustering: 50 Years Beyond K-Means," *Pattern Recognition Letters*, vol.31, no. 38, pp.651-666, June 2010 doi: 10.1016/j.patrec.2009.09.011

[43] Ensafi, R.; Dehghanzadeh, S.; Mohammad, R.; Akbarzadeh, T., "Optimizing Fuzzy K-means for network anomaly detection using PSO," *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on* , vol., no., pp.686,693, March 31 2008-April 4 2008 doi: 10.1109/AICCSA.2008.4493603,
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4493603&isnumber=4493499

[44] Yasami, Y.; Khorsandi, S.; Mozaffari, S.P.; Jalalian, A., "An unsupervised network anomaly detection approach by k-Means clustering & ID3 algorithms," *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on* , vol., no., pp.398,403, 6-9 July 2008 doi: 10.1109/ISCC.2008.4625717,
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4625717&isnumber=4625572

[45] Li Tian; Wang Jianwen, "Research on Network Intrusion Detection System Based on Improved K-means Clustering Algorithm," *Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum on* , vol.1, no., pp.76,79, 25-27 Dec. 2009 doi: 10.1109/IFCSTA.2009.25,

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5385128&isnumber=5385019

[46] Joglekar, S.P.; Tate, S.R., "ProtoMon: embedded monitors for cryptographic protocol intrusion detection and prevention," *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on* , vol.1, no., pp.81,88 Vol.1, 5-7 April 2004 doi: 10.1109/ITCC.2004.1286430,
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1286430&isnumber=28682

[47] Vik Tor Goh; Zimmermann, J.; Looi, M.; , "Towards Intrusion Detection for Encrypted Networks," *Availability, Reliability and Security, 2009. ARES '09. International Conference on* , vol., no., pp.540-545, 16-19 March 2009 doi: 10.1109/ARES.2009.76
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5066523&isnumber=5066394

[48] Vik Tor Goh; Zimmermann, J.; Looi, M.; , "Experimenting with an Intrusion Detection System for Encrypted Networks," *Int. J. of Business Intelligence and Data Mining* , vol. 5, no. 2, pp.172-191, January 2010 doi: 10.1504/IJBIDM.2010.031286

[49] Foroushani, V.A.; Adibnia, F.; Hojati, E.; , "Intrusion detection in encrypted accesses with SSH protocol to network public servers," *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on* , vol., no., pp.314-318, 13-15 May 2008

doi: 10.1109/ICCCE.2008.4580619

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4580619&isnumber=4580554

[50] Zhengjie Li; Yongzhong Li; Lei Xu, "Anomaly Intrusion Detection Method Based on K-Means Clustering Algorithm with Particle Swarm Optimization," *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on* , vol.2, no., pp.157,161, 24-25 Sept. 2011 doi: 10.1109/ICM.2011.184,

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6113492&isnumber=6113446

[51] (2013). *cyber espionage* [Online]. Available: http://lexicon.ft.com/Term?term=cyber-espionage

[52] (2013). *intrusion signature* [Online].

Available: http://www.webopedia.com/TERM/I/intrusion_signature.html

[53] (2013). *SCADA* [Online]. Available: http://www.webopedia.com/TERM/S/SCADA.html

[54] Tavallaee, M.; Bagheri, E.; Wei Lu; Ghorbani, A.A., "A detailed analysis of the KDD CUP 99 data set, "*Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on* , vol., no., pp.1,6, 8-10 July 2009

doi: 10.1109/CISDA.2009.5356528,

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5356528&isnumber=5356514

[55] Laheeb Ibrahim; Dujan Basheer; Mahmod Mahmod, "A COMPARISON STUDY FOR INTRUSION DATABASE (KDD99, NSL-KDD) BASED ON SELF ORGANIZATION MAP (SOM) ARTIFICIAL NEURAL NETWORK," *Journal of Engineering Science and Technology* , vol.8, no.1, pp.107-119, Feb. 2013

[56] (1999). *KDD Cup 1999: Computer network intrusion detection* [Online]. Available: http://www.kdd.org/kdd-cup-1999-computer-network-intrusion-detection

[57] (1999). *The NSL-KDD Data Set* [Online]. Available: http://nsl.cs.unb.ca/NSL-KDD/

[58] Hamerly, G.; Elkan, C., "Alternatives to the k-means algorithm that find better clusterings," *Proceedings of the eleventh international conference on Information and knowledge management (CIKM 2002)*, pp.600-607 doi: 10.1145/584792.584890, URL: http://doi.acm.org/10.1145/584792.584890

[59] K. V. Mardia et al., in *Multivariate Analysis*, 1st ed., Academic Press, 1980

[60] T. Menzies, A. Dekhtyar, J. Distefano and J. Greenwald, ""Problems with Precision: A Response to “Comments on ‘Data Mining Static Code Attributes to Learn Defect

Predictors’"," IEEE Transactions on Software Engineering, vol. 33, no. 9, pp. 637-640, 2007.

[61] N. Ailon, R. Jaiswal, C. Monteleoni. (2009). *Streaming k-means approximation* [Online]. Available: http://books.nips.cc/papers/files/nips22/NIPS2009_1085.pdf

[62] A. Meyerson, M. Shindler, A. Wong. (2011). *Fast and Accurate k-means for Large Datasets* [Online]. Available: http://books.nips.cc/papers/files/nips24/NIPS2011_1271.pdf

[63] D. Filimon. (2013). *Clustering data at scale* [Online]. Available: http://berlinbuzzwords.de/sites/berlinbuzzwords.de/files/slides/DanFilimon.pdf

[64] Erik Kangas. (2013, July 16). *SSL versus TLS - What's the difference?* [Online]. Available: http://luxsci.com/blog/ssl-versus-tls-whats-the-difference.html

[65] *TLS* [Online]. Available: http://www.webopedia.com/TERM/T/TLS.html

[66] Margaret Rouse. (2006, July). *Transport Layer Security (TLS)* [Online]. Available: http://searchsecurity.techtarget.com/definition/Transport-Layer-Security-TLS l

[67] Ariel Gilbert-Knight, Carlos Bergfeld, Adam Chapman. (2012, April 12). *An Introduction to Transport Layer Security* [Online]. Available: http://www.techsoup.org/support/articles-and-how-tos/introduction-to-transport-layer-security

[68] Margaret Rouse. (2007, March). *Secure Sockets Layer* [Online]. Available: http://searchsecurity.techtarget.com/definition/Secure-Sockets-Layer-SSL

[69] *SSL* [Online]. Available: http://www.webopedia.com/TERM/S/SSL.html

[70] *What is SSL (Secure Sockets Layer) and What Are SSL Certificates?* [Online]. Available: http://www.digicert.com/ssl.htm

[71] I. Perona, I. Gurrutxaga, O. Arbelaitz, J. I. Martin, J. Muguerza, J. M. Perez. (2008). *GureKddcup database description* [Online]. Available: http://www.sc.ehu.es/acwaldap/gureKddcup/README.pdf

[72] (2013). *TCP* [Online]. Available: http://www.webopedia.com/TERM/T/TCP.html

[73] A. A. Olusola et al., "Analysis of KDD'99 Intrusion Detection Dataset for Selection of Relevance Features," in Proceedings of The World Congress on Engineering and Computer Science, San Francisco, CA, 2010, pp. 162-168.

[74] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii., "Scalable k-means++," *Proc. VLDB Endow. (March 2012),* 5, 7, pp. 622-633. URL: http://dl.acm.org/citation.cfm?id=2180915

[75] Said, D.; Stirling, L.; Federolf, P.; Barker, K., "Data preprocessing for distance-based unsupervised Intrusion Detection," *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on* , vol., no., pp.181,188, 19-21 July 2011 doi: 10.1109/PST.2011.5971981,
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5971981&isnumber=5971950

Chapter 6: Appendices

Appendix A: Vocabulary Index

Title	Definition
Anomalies **(i.e. Outliers)**	Patterns in data that do not conform to a well-defined notion of normal behavior. [30] For example, a set of data points that is relatively different from the rest of the analyzed data set may be considered an anomaly.
Anomaly Detection **(i.e. Heuristic-based)**	Classifies traffic as normal or anomalous based on a set of heuristics, or rules. This technique attempts to detect misuse by monitoring a system's activities for any behavior that deviates from the norm. [16]
Address Resolution Protocol **(ARP)**	A protocol used in telecommunications in order to map a network layer address (i.e. IP Address) into a link layer address (i.e. MAC address).
Attack Toolkit	A hacker kit that exploits client-side vulnerabilities in order to execute arbitrary code. [32]
Botnets **(i.e. Bot Networks)**	A network of compromised computers, known as drones or zombies, that are used by cyber criminals in order to transmit spam messages, spread malware, and/or for other criminal activities. [32]
Clustering Criterion	The mathematical formulation of the inductive principle. This criterion is used to differentiate various clustering models given the same data set. [35]
Clustering Model	A structure used to represent a cluster. [35]
Cyberespionage **(i.e. Cyber Espionage or**	The act or practice of obtaining confidential information, stored in digital formats on computers or IT networks, without the permission of

Cyber Spying)	the holder through the use of cracking techniques or malicious software. [51]
Dynamic Clustering Algorithm	This algorithm formulates the clustering problem in terms of the Maximum-Likelihood Estimation framework and allows the use of representations other than the centroid for each cluster. [36]
False Positive (i.e. Type I Error)	A false positive is a sequence of innocuous events that an IDS erroneously classifies as intrusive [2][15].
False Positive Rate (i.e. False Alarm Rate)	The expectancy of producing a false positive.
False Negative (i.e. Type II Error)	A false negative is sequence of unwanted traffic or intrusion attempts that an IDS fails to detect or report. [2][15]
False Negative Rate	The expectancy of producing a false negative.
Host-based	The data, or set of packets, to and from a single host is used to detect signs of an intrusion. [16]
Inductive Principle (i.e. Induction Principle)	A mathematical formalization for the definition of a *cluster*. This principle is used to make explicit a clustering criterion, in order to select a "best fit" structure given a set of data [35]
Intrusion Detection	Intrusion Detection the process of monitoring a network or system for potential signs of malicious activities or policy violations.
Intrusion Detection System (i.e. IDS)	An intrusion detection system is a device that attempts to detect an intrusion into a network or system using observed information and/or audit data. It can be a piece of installed software or a physical component that monitors traffic in order to detect unwanted activities, events, and/or policy violations. [2]

Intrusion Prevention System (i.e. IPS)	An intrusion prevention system is a device that attempts to both detect and prevent an intrusion. [2]
Misuse Detection (i.e. Signature-based)	Classifies based on patterns or signatures. This detection technique can only detect an intrusion for which a signature already exists. [16]
Network-based	The data from a network is scrutinized against a database or model in order to flag packets that are potentially malicious. Audit data from one or multiple hosts may also be incorporated to detect signs of an intrusion. [16]
Particle Swarm Optimization Algorithm (i.e. PSOA)	An algorithm from the field of swarm intelligence. This algorithm was first introduced as a substitute for a genetic algorithm (i.e. GA). It operates on the basis that consecutive actions of respective individuals are influenced by their own movements and those of their companions. [38]
Secure Sockets Layer (i.e. SSL)	A commonly-used protocol for managing the security of a message transmission between a server and a client. [68-69] It has been succeeded by Transport Layer Security (TLS), which is based on SSL. [70]
Signature (i.e. Digital Fingerprint; Digital Footprint)	Recorded evidence of a system intrusion (i.e. digital footprint). For example, the number of failed logins or the unauthorized execution of software may be an example of a signature. [52]
Supervisory Control And Data Acquisition Systems (i.e. SCADA)	A type of industrial control system used for gathering and analyzing real-time data or information. Usually, they have been used to monitor and control industry equipment or plant infrastructures. (e.g. telecommunications, water and waste control, energy, oil, etc) [53]
Transmission Control	One of the main protocols in TCP/IP networks. TCP enables two hosts to

Protocol (i.e. TCP)	establish a connection and exchange streams of data. It also provides a guarantee for the delivery of the data as well as maintaining the order in which the data packets were sent. [72]
Transport Layer Security (i.e. TLS)	A protocol that provides data encryption and authentication between applications and servers. [64] It is based on Netscape's SSL 3.0 protocol [65-66], and is considered to be the successor of SSL. [67] While the differences between TLS and SSL are minor and very technical [67], they are not interoperable.[65-66]
Zero-Day Vulnerabilities	A vulnerability that is unknown to others, undisclosed to the software developer, or for which no security fix is available. [32]

Appendix B: Java Implementation

Appendix B.1: K-Means Clustering Algorithm

```
/*
 * Name: Luis C. Armendariz
 * Program Name: K-Means Clustering Algorithm
 * Advisor: Dr. Roy S. Nutter
 * Date: 10/25/2013
 * Program Description: Parses KDD & NSL-KDD datasets from input files and performs K-Means
Clustering on SSH data
*/

import java.util.Scanner;                //Get scanner to parse input file into pieces
import java.io.FileReader;               //Get file reader to read input file as whole
import java.io.FileNotFoundException;    //Error Handling
import java.util.Arrays;                 //Gets static methods for Arrays
import java.util.ArrayList;              //Import ArrayList
import java.util.Random;                 //Import Random for random # generator
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;
import java.math.BigDecimal;             //Allows rounding of double type values

public class kMeans
{
 static Scanner console = new Scanner(System.in);   //Initialize scanner for reading user input

 public static void main(String[] args) throws FileNotFoundException
 {
  /*********************************************
   ************ DATA INITIALIZATION ************
   *********************************************/

  //Initialize run time
  long startTime = System.currentTimeMillis();
  String runTime = "";

  //Initialize variables
  String inFileName = "KDDTrain+";   //"kddcup.data"; //KDDTrain+";   //Input file name (w/o .txt)
  String delimiter = "\\,";         //Delimiter token (comma)
  String connType = "ssh";          //Type of connection to examine
  boolean normCheck = true;         //Check to normalize input data from dataset

  int[] fitPattern;                //Pattern for continuous data locations
  int labelDistance;               //Pattern label distance to factor during fitness
```

```
   String inLine = "";              //Input line of data
   String normLine = "";            //Input line of normalized data
   ArrayList<String> volume;        //Volume of data values (Array List of Strings)
   ArrayList<String> volumeNorm;    //Volume of normalized data values
   ArrayList<String> volumeOrig;    //Volume of original data values
   int totalPatterns = 0;           //Total # of Patterns

   int k = 0;                       //Number of cluster centers (i.e. centroids)
   String randomPattern = "";       //Random pattern
   ArrayList<Cluster> clusterList;  //List of clusters
   ArrayList<Integer> patternGuesses; //List of pattern selections
   Random generator;                //Random # generator
   int randomNum;                   //Random Number

   //Initialize Thread Pool Manager
   ExecutorService es = Executors.newCachedThreadPool();

   int[] clusterAssign;             //Keyed Array of Pattern to Cluster assignments [Ex: Pattern1=>Cluster5]
   AtomicInteger clusterReassigns;  //Number of cluster reassignments
   int clusterLoops = 0;            //Number of clustering loops


   //Get input file
   Scanner inFile = new Scanner (new FileReader (inFileName + ".txt"));     //Get input file

   //Set fitness pattern & label distance (for continous data analysis)
   switch(inFileName)
   {
    //NSL-KDD
    case "KDDTrain+":
     //fitPattern = new
int[]{0,4,5,7,8,9,10,12,13,14,15,16,17,18,19,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40};
//All continuous elements
      fitPattern = new int[]{0,4,5,7,8};  //Only continuous TCP elements
      labelDistance = 2;
      break;
    //KDD
    case "kddcup.data":
     //fitPattern = new
int[]{0,4,5,7,8,9,10,12,13,14,15,16,17,18,19,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40};
//All continuous elements
      fitPattern = new int[]{0,4,5,7,8};
      labelDistance = 1;
      break;
    case "kddcup.data_10_percent":
     //fitPattern = new
int[]{0,4,5,7,8,9,10,12,13,14,15,16,17,18,19,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40};
      fitPattern = new int[]{0,4,5,7,8};
```

```
    labelDistance = 1;
    break;
  default:
    fitPattern = new int[0];
    labelDistance = 1;
    break;
}


//Traverse input file for SSH data
//volume = new ArrayList<String>();
volumeNorm = new ArrayList<String>();
volumeOrig = new ArrayList<String>();
while (inFile.hasNext())
{
  //Get next line of data
  inLine = inFile.nextLine();

  //Check if line contains SSH data
  if(inLine.toLowerCase().contains(connType.toLowerCase()))
  {
    //Check to normalize the data
    if(normCheck)
    {
      //Step C: If found, Normalize the data using Log Normalization
      normLine = logNormalize(inLine, delimiter, fitPattern);
      //System.out.println("In:   " + inLine);
      //System.out.println("Norm: " + normLine);

      //Add normalized SSH data to end of normalized volume
      volumeNorm.add(normLine);
    }
    else
    {
      //Add original SSH data to end of volume
      volumeOrig.add(inLine);
    }
  }
}

//Set volume for clustering (check for normalization)
if(normCheck)
{
  volume = volumeNorm;
}
else
{
  volume = volumeOrig;
```

```
}

//Set total number of patterns
totalPatterns = volume.size();



/**********************************************
 ********** K-Means Clustering **********
 **********************************************/

/**********************************************
 ****************** STEP #1 ******************
 **********************************************/
/* Instructions
 * -----------
 * Choose k cluster centers to coincide with k
 * randomly-chosen patterns or k randomly defined
 * points inside the hyper volume containing the
 * pattern set
 */


//Set value of k
k = (int) Math.sqrt(totalPatterns/2);   //Rule of Thumb by Mardia et al.[59]

//Setup list variables and random number generator
clusterList = new ArrayList<Cluster>();
patternGuesses = new ArrayList<Integer>();
generator = new Random();

//Choose k random patterns from volume to make k cluster centers
for(int i=0; i < k; i++)
{
 //Get a pattern from volume
 try
 {
  //Get a random unchosen pattern location
  do
  {
   //Generate a "random" number from 0 to totalPatterns-1
   randomNum = generator.nextInt(totalPatterns);

   //Check if number was already chosen  (prevents selecting the same patterns as centroids)
  }while(patternGuesses.contains(randomNum));

  //Add pattern location to list
```

```
      patternGuesses.add(randomNum);

      //Get pattern
      randomPattern = volume.get(randomNum);
     }
     catch(IndexOutOfBoundsException err)
     {
      System.err.println("An error occurred: " + err.getMessage());
     }

     //Create cluster & assign centroid (Split pattern into element pieces)
     Cluster cluster = new Cluster(randomPattern.split(delimiter), labelDistance, fitPattern);

     //Add cluster to Cluster ArrayList
     clusterList.add(cluster);
    }


    /***** Prepare for Looping *****/
    //Initialize cluster assignment array
    clusterAssign = new int[totalPatterns];



    //Begin K-Means Clustering Loop
    do
    {
     /************************************************
      ******************** STEP #2 *******************
      ************************************************/
     /* Instructions
      * ------------
      * Assign each pattern to closest cluster center
      */

     //Reset cluster reassignment count
     clusterReassigns = new AtomicInteger(0);

     //Initialize Thread Pool Manager (for RunnableCluster Threads)
     es = Executors.newCachedThreadPool();

     //Loop through volume
     for(int i=0; i < totalPatterns; i++)
     {
      //Create RunnableCluster Object
      RunnableCluster threadAssign = new RunnableCluster(i, volume, clusterList, clusterAssign,
clusterLoops, clusterReassigns, delimiter);
```

```
  //Execute thread to assign pattern to closest cluster centroid
  es.execute(threadAssign);
}

//Shutdown thread pool & finish all queued threads
shutdownAndAwaitTermination(es);


//System.out.println("2 Done");

/************************************************
  ******************** STEP #3 ********************
  ************************************************/
/* Instructions
 * ------------
 * Recompute the cluster centers
 * using the current cluster memberships
 */

//Initialize Thread Pool Manager (for RunnableCentroid Threads)
es = Executors.newCachedThreadPool();

//Loop through cluster list
for(int i=0, j=clusterList.size(); i < j; i++)
{
  //Create RunnableCentroid Object
  RunnableCentroid threadUpdate = new RunnableCentroid(clusterList.get(i), volume, delimiter);

  //Execute thread to recompute cluster centroid
  es.execute(threadUpdate);
}

//Shutdown thread pool & finish all queued threads
shutdownAndAwaitTermination(es);


//System.out.println("3 Done");

/************************************************
  ******************** STEP #4 ********************
  ************************************************/
/* Instructions
 * ------------
 * If a convergence criterion is not met, go to step 2
 *
 * Typical convergence criteria are:
 * * no (or minimal) reassignment of patterns to new cluster centers,
```

```
     * * or minimal decrease in squared error
     */

     clusterLoops = clusterLoops + 1;

     //System.out.println("4 Done = Reassigns: " + clusterReassigns.get() + " Ratio: " + (totalPatterns/16));

   }while(clusterReassigns.get() > (totalPatterns/8));



   /************************************************
    ***************** IDS Analysis *****************
    ************************************************/
   /* Instructions
    * -----------
    * Uses Traffic Properties to Identify Centroids with Attack Traffic (High Level Analysis)
    *
    * Future Work:
    * Scan each cluster pattern and make best attempts to identify individual attacks (Deep Level
Analysis)
    */

   //Initialize array to hold analysis results
   String[] ids_results = new String[clusterList.size()];

   //Loop through cluster list
   for(int i=0, j=clusterList.size(); i < j; i++)
   {
     //Analyze each cluster for probability of attacks
     ids_results[i] = ids_analyze(clusterList.get(i), normCheck);
   }



   /************************************************
     ************** ENDING CLEAN UP **************
     ************************************************/

   //Calculate run time
   runTime = getRunTime(startTime);

   //Completion message
   System.out.println("The program has completed execution.");
   System.out.println();
   System.out.println("File Name: " + inFileName);
```

```
  System.out.println("Total Run Time: " + runTime);
  System.out.println("Total Patterns: " + totalPatterns);
  System.out.println("Total Loops: " + clusterLoops);
  System.out.println("K: " + k);
  System.out.println();

  //printClusterListPatterns(clusterList);
  //System.out.println();
  printClusterListPatternsWithLabelsAsSummary(clusterList,volume,delimiter);
  System.out.println();
  //printClusterListCentroids(clusterList);
  //System.out.println();
  printIDSAnalysis(ids_results);

}//end main




/*******************************************
 ************** Function List **************
 *******************************************/

private static String getRunTime(long sTime)
{
  //Calculate run time
  long endTime   = System.currentTimeMillis();
  long totalTime = endTime - sTime;
  String runTime = millisecondsToStr(totalTime);

  return runTime;
}//end getRunTime

private static String ids_analyze(Cluster cluster, boolean normCheck)
{
  String retStr = "";                       //return value

  //Get cluster centroid
  String[] centroid = cluster.getCentroid();

  //System.out.print("Centroid: ");
  //cluster.printCentroid();
  //System.out.println();

  //Check if cluster is empty
  if(cluster.getPatternListSize() == 0)
  {
```

```
    retStr =  "(empty)";
   }
   else
   {
    /* Analyze cluster centroid based on traffic properties */
    //Analyze Flow Sizes [Based on Foroushani et al.[49] & NSL/KDD Datasets]
    double src_size_dec = Double.valueOf(centroid[4]);  //number of data bytes from source to
destination
    double dst_size_dec = Double.valueOf(centroid[5]);  //number of data bytes from destination to
source
    double factor = 0;                          //factor ratio of source to dest. bytes

    //Check for log normalization
    if(normCheck)
    {
     //Denormalize values
     src_size_dec = (Math.ceil(Math.pow(10,src_size_dec)) - 1);
     dst_size_dec = (Math.ceil(Math.pow(10,src_size_dec)) - 1);
    }

    //Round values up to nearest whole number
    float src_size = (float)Math.ceil(src_size_dec);
    float dst_size = (float)Math.ceil(dst_size_dec);

    //Get factor value
    if((src_size != 0))
    {
     factor = dst_size / src_size;
     //System.out.println("Src Size: " + src_size + " Dst Size: " + dst_size + " Factor: " + factor);
    }

    //If small request
    if(src_size <= dst_size)
    {
     //With big response, classify Normal
     if(factor > 10)
     {
      retStr = "Normal";
      //System.out.println();
      //cluster.printPatternListExpanded(volume, delimiter);
      //System.out.println();
     }

     //With small response, classify Anomalous (e.g. Scanning Attack (Probe))
     else
     {
      retStr = "Anomalous";
     }
```

```
    }

    //If big request with smaller response, classify Anomalous (e.g. Buffer Overflow)
    else
    {
     retStr = "Anomalous";
    }
  }
  return retStr;
}//end ids_analyze

private static boolean isNumeric(String str)
{
  return str.matches("-?\\d+(\\.\\d+)?");  //match a number (latin digits) with optional '-' and decimal.
}//end isNumeric [Open Source]

public static String logNormalize(String inLine, String delimiter, int[] fitPattern)
{
  //Declare variables
  String inPiece = "";
  String normLine = "";

  //Split input line into pieces
  String[] inLinePieces = inLine.split(delimiter);

  //For each fitPattern value
  for(int i=0,j=fitPattern.length; i < j; i++)
  {
    //Get corresponding input line piece
    inPiece = inLinePieces[fitPattern[i]];

    //Check if piece is continuous (i.e. numeric)
    if(isNumeric(inPiece))
    {
      //If numeric, convert from string to numeric (Xij)
      double inData = Double.valueOf(inPiece);

      //Apply log normalization formula (X'ij = log(1 + Xij))
      inData = Math.log10(1 + inData);

      //Round value to 2 decimal places
      inData = round(inData, 2, BigDecimal.ROUND_HALF_UP);

      //Store new value in piece location
      inPiece = String.valueOf(inData);

      //Store normalized piece back in array of pieces
      inLinePieces[fitPattern[i]] = inPiece;
```

```
  }
 }

 //Create string from array of strings
 StringBuilder sb = new StringBuilder();

 for(String s: inLinePieces)
 {
  sb.append(s).append(',');
 }

 sb.deleteCharAt(sb.length()-1); //delete last comma

 normLine = sb.toString();

 //Return normalized input line as complete string
 return normLine;
}//end logNormalize


public static String millisecondsToStr(long milliseconds)
{
 // TIP: to find current time in milliseconds, use:
 // var  current_time_milliseconds = new Date().getTime();

 // This function does not deal with leap years, however,
 // it should not be an issue because the output is approximated

 String retStr = "";
 double temp;

 //Convert ms to seconds (if applicable)
 if ((milliseconds / 1000) != 0)
 {
  temp = milliseconds / 1000;

  //Get years
  double years = Math.floor(temp / 31536000);
  if (years != 0)
  {
   temp %= 31536000;
   retStr += years + " Year" + numberEnding(years) + " ";
  }

  //Get days
  double days = Math.floor(temp / 86400);
  if (days != 0)
  {
```

```
    temp %= 86400;
    retStr += days + " Day" + numberEnding(days) + " ";
   }

   //Get hours
   double hours = Math.floor(temp / 3600);
   if (hours != 0)
   {
    temp %= 3600;
    retStr += hours + " hr" + numberEnding(hours) + " ";
   }

   //Get minutes
   double minutes = Math.floor(temp / 60);
   if (minutes != 0)
   {
    temp %= 60;
    retStr += minutes + " min" + numberEnding(minutes) + " ";
   }

   //Get seconds
   double seconds = temp;
   if (seconds != 0)
   {
    retStr += seconds + " sec" + numberEnding(seconds) + " ";
   }

   //Get milliseconds
   double milliSec = milliseconds % 1000;
   if (milliSec != 0)
   {
    retStr += milliSec + " ms ";
   }
  }
  else
  {
     retStr += milliseconds + " ms";
  }

  return retStr;
}//end millisecondsToStr [Open Source]

private static String numberEnding(double numb)
{
  String retVal;

  if(numb > 1)
  {
```

```
    retVal = "s";
   }
   else
   {
    retVal = "";
   }

   return retVal;
  }//end numberEnding

  private static void printClusterAssigns(int[] clusterAssignments)
  {
   for(int r=0,s=clusterAssignments.length; r < s; r++)
   {
    System.out.print(r + "->" + clusterAssignments[r] + " ");
   }
   System.out.println();
  }//end printClusterAssigns

  private static void printClusterListCentroids(ArrayList<Cluster> clusterL)
  {
   for(int i=0,j=clusterL.size(); i < j; i++)
   {
    System.out.println(i + " " + Arrays.toString(clusterL.get(i).getCentroid()));
   }
  }//end printClusterListCentroids

  private static void printClusterListPatterns(ArrayList<Cluster> clusterL)
  {
   for(int i=0,j=clusterL.size(); i < j; i++)
   {
    System.out.print("Cluster " + i + ": ");
    clusterL.get(i).printPatternList();
   }
  }//end printClusterListPatterns

  private static void printClusterListPatternsWithLabels(ArrayList<Cluster> clusterL, ArrayList<String>
volume, String delimiter)
  {
   for(int i=0,j=clusterL.size(); i < j; i++)
   {
    System.out.print("Cluster " + i + ": ");
    clusterL.get(i).printPatternListWithLabels(volume, delimiter);
   }
  }//end printClusterListPatternsWithLabels

  private static void printClusterListPatternsWithLabelsAsSummary(ArrayList<Cluster> clusterL,
ArrayList<String> volume, String delimiter)
```

```
{
  for(int i=0,j=clusterL.size(); i < j; i++)
  {
    System.out.print("Cluster " + i + ": ");
    clusterL.get(i).printPatternListWithLabelsAsSummary(volume, delimiter);
  }
}//end printClusterListPatternsWithLabelsAsSummary

private static void printIDSAnalysis(String[] ids_results)
{
  for(int i=0,j=ids_results.length; i < j; i++)
  {
    System.out.println("Cluster " + i + ": " + ids_results[i]);
  }
}//end printIDSAnalysis

public static double round(double unrounded, int precision, int roundingMode)
{
  BigDecimal bd = new BigDecimal(unrounded);
  BigDecimal rounded = bd.setScale(precision, roundingMode);
  return rounded.doubleValue();
}//end round [Open Source]

private static void shutdownAndAwaitTermination(ExecutorService pool)
{
  //Disable new tasks from being submitted & finish all existing threads in queue
  pool.shutdown();

  try
  {
    //Wait a while for existing tasks to terminate
    if(!pool.awaitTermination(1, TimeUnit.MINUTES))
    {
      pool.shutdownNow();  //Cancel currently executing tasks

      //Wait a while for tasks to respond to being cancelled
      if(!pool.awaitTermination(1, TimeUnit.MINUTES))
      {
        System.err.println("Pool did not terminate");
      }
    }
  }
  catch (InterruptedException ie)
  {
    System.err.println("An error occurred: " + ie.getMessage());

    // (Re-)Cancel if current thread also interrupted
    pool.shutdownNow();
```

```
      //Preserve interrupt status
      Thread.currentThread().interrupt();
    }
  }//end shutdownAndAwaitTermination [Open Source]
}//end kMeans
```

Appendix B.2: K-Means Cluster class

```
/*
 * Name: Luis C. Armendariz
 * Program Name: Cluster
 * Advisor: Dr. Roy S. Nutter
 * Date: 10/29/2013
 * Program Description: Creates a Cluster class for the K-Means Algorithm
*/

import java.util.ArrayList;              //Import ArrayList
import java.util.Iterator;               //Allows traversal of an ArrayList
import java.util.Map;                    //Get methods for Map
import java.util.HashMap;                  //Import HashMap

public class Cluster
{
 private ArrayList<Integer> patternList;  //List of pattern locations
 private String[] centroid;             //Cluster center
 private final int labelDistance;        //Pattern label distance for fitness
 private final int[] fitPattern;        //Pattern for continuous data locations
 private Object lock1 = new Object();    //Lock 1 for synchronization with threads
 private Object lock2 = new Object();    //Lock 2 for synchronization with threads
 private Object lock3 = new Object();    //Lock 3 for synchronization with threads

 /********************/
 /*** Constructors ***/
 /********************/
 public Cluster(int totalElements, int labelDistance, int[] fitPattern)
 {
  this.patternList = new ArrayList<Integer>();
  this.centroid = new String[totalElements];
  this.labelDistance = labelDistance;
  this.fitPattern = fitPattern;
 }//end Cluster

 public Cluster(String[] inCentroid, int labelDistance, int[] fitPattern)
 {
  this.patternList = new ArrayList<Integer>();
  this.centroid = inCentroid;
  this.labelDistance = labelDistance;
  this.fitPattern = fitPattern;
 }//end Cluster(String)



 /********************/
 /**** Accessors *****/
```

```
 /********************/
 public String[] getCentroid()
 {
  return centroid;
 }//end getCentroid

 public double getFitness(String[] pattern)
 {
  //Initialize variables
  double fitValue = 0;
  double sumSquare = 0.0;

  //Euclidean Distance Function = d(p,q) = sqrt(sumFromI=1ToN((qi - pi)^2))

  //Check for fitness pattern
  if(fitPattern.length > 0)
  {
   //For length of fitness pattern, get Continuous Data elements
   for(int i=0,j=fitPattern.length; i < j; i++)
   {
    //Get an element
    String strData = pattern[fitPattern[i]];

    //Check if element is numerical value
    if(isNumeric(strData))
    {
     //If numeric, convert from string to numeric (q)
     double inData = Double.valueOf(strData);

     //Take the difference between the Mean element (q-p)
     double diff = inData - Double.valueOf(centroid[fitPattern[i]]);

     //Square the difference (q-p)^2
     double square = Math.pow(diff,2);

     //Add value to total sum of squares E(qi-pi)^2
     sumSquare = sumSquare + square;
    }
   }
  }
  else
  {
   //For length of String array, get only Numerics
   for(int i=0,j=pattern.length - labelDistance; i < j; i++)   //Pattern.length-2 is to avoid factoring in the
attack labels as part of fitness value
   {
    //Get an element
    String strData = pattern[i];
```

```
    //Check if element is numerical value
    if(isNumeric(strData))
    {
     //If numeric, convert from string to numeric (q)
     double inData = Double.valueOf(strData);

     //Take the difference between the Mean element (q-p)
     double diff = inData - Double.valueOf(centroid[i]);

     //Square the difference (q-p)^2
     double square = Math.pow(diff,2);

     //Add value to total sum of squares E(qi-pi)^2
     sumSquare = sumSquare + square;
    }
   }
  }

  //Take square root of sum of squares to find Euclidean Distance
  fitValue = Math.sqrt(sumSquare);

  return fitValue;
}//end getFitness

public int[] getFitPattern()
{
  return fitPattern;
}//end getFitPattern

public int getLabelDistance()
{
  return labelDistance;
}//end getLabelDistance

public Integer getPattern(int location)
{
  Integer outPattern = 0;

  try
  {
   outPattern = patternList.get(location);
  }
  catch(IndexOutOfBoundsException err)
  {
   System.out.println("An error occurred: " + err.getMessage());
  }
```

```
  return outPattern;
}//end getPattern

public ArrayList<Integer> getPatternList()
{
  return patternList;
}//end getPatternList

public int getPatternListSize()
{
  return patternList.size();
}//end getPatternListSize



/*******************/
/***** Mutators *****/
/*******************/
public void addPattern(Integer pattern)
{
  synchronized(lock1)
  {
    patternList.add(pattern);
  }
}//end addPattern

public void removePattern(Integer pattern)
{
  synchronized(lock2)
  {
    patternList.remove(pattern);  //Remove Integer Object of unique pattern location w.r.t. volume
  }
}//end removePattern

public void setPattern(Integer pattern, int location)
{
  synchronized(lock3)
  {
    patternList.set(location, pattern);
  }
}//end setPattern



/*******************/
/**** Executors *****/
/*******************/
public void printCentroid()
```

```
{
  for(int i=0,j=centroid.length; i < j ; i++)
  {
    System.out.print(centroid[i]);

    if((i+1) != j)
    {
      System.out.print(",");
    }//end if
  }//end for
}//end printCentroid

public void printPatternList()
{
  Iterator<Integer> it = patternList.iterator();    //Attach patternList to iterator for traversal

  if(it.hasNext())
  {
    while(it.hasNext())
    {
      System.out.print(it.next() + " " );
    }
    System.out.println();
  }
  else
  {
    System.out.println("(empty)");
  }
}//end printPatternList

public void printPatternListExpanded(ArrayList<String> volume, String delimiter)
{
  Iterator<Integer> it = patternList.iterator();    //Attach patternList to iterator for traversal

  //Check for patterns
  if(it.hasNext())
  {
    //Loop through patterns
    while(it.hasNext())
    {
      //Get a pattern from volume
      String[] tempPat = volume.get(it.next()).split(delimiter);

      //Print entire pattern
      /*
      for(int i=0,j=tempPat.length; i < j; i++)
      {
        System.out.print(tempPat[i]);
```

```
      if((i+1) != j)
      {
        System.out.print(",");
      }
     }
     System.out.println();
     */

     //Print Src_bytes and Dst_bytes elements of pattern
     System.out.println(tempPat[4] + "," + tempPat[5]);
   }
   System.out.println();
  }
  else
  {
   System.out.println("(empty)");
  }
}//end printPatternListExpanded

public void printPatternListWithLabels(ArrayList<String> volume, String delimiter)
{
  Iterator<Integer> it = patternList.iterator();    //Attach patternList to iterator for traversal

  if(it.hasNext())
  {
    while(it.hasNext())
    {
      String[] tempPat = volume.get(it.next()).split(delimiter);

      System.out.print(translate(tempPat[tempPat.length - labelDistance]) + " " );
    }
    System.out.println();
  }
  else
  {
    System.out.println("(empty)");
  }
}//end printPatternListWithLabels

public void printPatternListWithLabelsAsSummary(ArrayList<String> volume, String delimiter)
{
  Iterator<Integer> it = patternList.iterator();    //Attach patternList to iterator for traversal
  Map<String, Integer> map = new HashMap<String, Integer>(); //Summary of attack types

  if(it.hasNext())
  {
    while(it.hasNext())
```

```
   {
    //Get pattern split into pieces
    String[] tempPat = volume.get(it.next()).split(delimiter);

    //Get attack type
    String attackType = translate(tempPat[tempPat.length - labelDistance]);

    //Add attack type to hashed array
    if(map.containsKey(attackType))
    {
     map.put(attackType, map.get(attackType) + 1);
    }
    else
    {
     map.put(attackType, 1);
    }
   }

   //Print out summaries of all attack types
   for(String name: map.keySet())
   {
    String key = name.toString();
    String value = map.get(name).toString();
    System.out.print(key + " x" + value + " ");
   }

   System.out.println();
  }
  else
  {
   System.out.println("(empty)");
  }
}//end printPatternListWithLabelsAsSummary



/************************/
/*** Private Functions ***/
/************************/
private static boolean isNumeric(String str)
{
  return str.matches("-?\\d+(\\.\\d+)?");  //match a number (latin digits) with optional '-' and decimal.
}//end isNumeric [Open Source]

private static String translate(String str)
{
  String retStr = "";
```

```
    //Remove ending period (if necessary)
    str = str.replace(".", "");

    //Check string
    switch(str)
    {
      case "back": retStr = "dos"; break;
      case "buffer_overflow": retStr = "u2r"; break;
      case "ftp_write": retStr = "r2l"; break;
      case "guess_passwd": retStr = "r2l"; break;
      case "imap": retStr = "r2l"; break;
      case "ipsweep": retStr = "probe"; break;
      case "land": retStr = "dos"; break;
      case "loadmodule": retStr = "u2r"; break;
      case "multihop": retStr = "r2l"; break;
      case "neptune": retStr = "dos"; break;
      case "nmap": retStr = "probe"; break;
      case "perl": retStr = "u2r"; break;
      case "phf": retStr = "r2l"; break;
      case "pod": retStr = "dos"; break;
      case "portsweep": retStr = "probe"; break;
      case "rootkit": retStr = "u2r"; break;
      case "satan": retStr = "probe"; break;
      case "smurf": retStr = "dos"; break;
      case "spy": retStr = "r2l"; break;
      case "teardrop": retStr = "dos"; break;
      case "warezclient": retStr = "r2l"; break;
      case "warezmaster": retStr = "r2l"; break;
      case "normal": retStr = "normal"; break;
      default: retStr = "str"; break;
    }

    return retStr;
  }//end translate
}//end Cluster
```

Appendix B.3: RunnableCluster class

```
/*
 * Name: Luis C. Armendariz
 * Program Name: RunnableCluster
 * Advisor: Dr. Roy S. Nutter
 * Date: 11/4/2013
 * Program Description: Implements the Runnable interface for assigning patterns to a K-Means cluster
*/

import java.util.ArrayList;               //Import ArrayList
import java.util.concurrent.atomic.AtomicInteger;

public class RunnableCluster implements Runnable
{
  private final int patternLoc;           //Pattern location in volume
  private final ArrayList<String> volume;
  private final ArrayList<Cluster> clusterList;
  private final int[] clusterAssign;
  private final int clusterLoops;
  private final AtomicInteger clusterReassigns;
  private final String delimiter;

  /*********************/
  /*** Constructors ***/
  /*********************/
  public RunnableCluster(int patternLocation, ArrayList<String> volume, ArrayList<Cluster> clusterList,
int[] clusterAssign, int clusterLoops, AtomicInteger clusterReassigns, String delimiter)
  {
    this.patternLoc = patternLocation;
    this.volume = volume;
    this.clusterList = clusterList;
    this.clusterAssign = clusterAssign;
    this.clusterLoops = clusterLoops;
    this.clusterReassigns = clusterReassigns;
    this.delimiter = delimiter;
  }//end RunnableCluster


  /*********************/
  /**** Executors *****/
  /*********************/
  public void run()
  {
    //Initialize variables
    String tempPattern = "";          //Temporary Pattern (Whole)
    String[] dataPattern;             //Temporary Pattern (Split into data elements)
    Double tempFitness = 0.0;         //Temporary Fitness Value
```

```
Double bestFitness = -1.0;        //Best Fitness Value for a temporary pattern
int bestCluster = -1;             //Best Cluster for temporary pattern

//Get pattern
tempPattern = volume.get(patternLoc);

//Split pattern using delimiter
dataPattern = tempPattern.split(delimiter);

//Get fitness with 1st cluster
bestFitness = clusterList.get(0).getFitness(dataPattern);
bestCluster = 0;

//Check fitness with other clusters and take lowest
for(int j=1, m=clusterList.size(); j < m; j++)
{
  //Get fitness of cluster j
  tempFitness = clusterList.get(j).getFitness(dataPattern);

  //Best fitness is LOWEST of values
  if(tempFitness < bestFitness)
  {
    bestFitness = tempFitness;
    bestCluster = j;
  }
}


//Check cluster assignment
if(clusterLoops == 0)
{
  //Typecast pattern location to Object
  Integer pattern = (Integer) patternLoc;

  //Assign pattern to cluster with best fitness
  clusterList.get(bestCluster).addPattern(pattern);

  //Store cluster assignment
  clusterAssign[patternLoc] = bestCluster;

  //Increment reassignment count
  clusterReassigns.incrementAndGet();
}
else if(clusterAssign[patternLoc] != bestCluster)
{
  //Typecast pattern location to Object
  Integer pattern = (Integer) patternLoc;
```

```
    //Remove pattern from old cluster (if value exists)
    clusterList.get(clusterAssign[patternLoc]).removePattern(pattern);

    //Assign pattern to cluster with best fitness
    clusterList.get(bestCluster).addPattern(pattern);

    //Store cluster assignment
    clusterAssign[patternLoc] = bestCluster;

    //Increment reassignment count
    clusterReassigns.incrementAndGet();
   }//end if-else
  }//end run
}//end RunnableCluster
```

Appendix B.4: RunnableCentroid class

```
/*
 * Name: Luis C. Armendariz
 * Program Name: RunnableCentroid
 * Advisor: Dr. Roy S. Nutter
 * Date: 11/3/2013
 * Program Description: Implements the Runnable interface for updating a K-Means cluster centroid
*/

import java.util.ArrayList;              //Import ArrayList
import java.math.BigDecimal;             //Allows rounding of double type values

public class RunnableCentroid implements Runnable
{
 private final Cluster cluster;
 private final ArrayList<String> volume;
 private final String delimiter;

 /*********************/
 /*** Constructors ***/
 /*********************/
 public RunnableCentroid(Cluster inCluster, ArrayList<String> inPatterns, String delim)
 {
  this.cluster = inCluster;
  this.volume = inPatterns;
  this.delimiter = delim;
 }//end RunnableCentroid



 /*********************/
 /**** Executors *****/
 /*********************/
 public void run()
 {
  String[] tempPattern;
  ArrayList<Integer> patternList = cluster.getPatternList();
  int totalPatterns = cluster.getPatternListSize();
  String[] centroid = cluster.getCentroid();
  int labelDistance = cluster.getLabelDistance();
  int[] fitPattern = cluster.getFitPattern();

  //Check for fitness pattern (continuous data values)
  if(fitPattern.length > 0)
  {
   /* PATTERN VALUE SUMATION */
   //Loop through list of patterns
```

```
for(int i=0,j=totalPatterns; i < j; i++)
{
  //Get a pattern and split into pieces
  tempPattern = (volume.get(patternList.get(i))).split(delimiter);

  //Loop through continuous data elements
  for(int m=0,n=fitPattern.length; m < n; m++)
  {
    //Get location of continuous element
    int cLoc = fitPattern[m];

    //If numerical values
    if(isNumeric(centroid[cLoc]) && isNumeric(tempPattern[cLoc]))
    {
      //Get numerical value(s)
      double cValue = Double.valueOf(centroid[cLoc]);
      double pValue = Double.valueOf(tempPattern[cLoc]);

      //Sum each NUMERICAL pattern value with respective centroid value
      if(i == 0)
      {
        cValue = pValue;  //For 1st pattern, assign pattern element to centroid element
      }
      else
      {
        cValue = cValue + pValue;
      }

      //Convert back to a string & update centroid value
      centroid[cLoc] = String.valueOf(cValue);
    }
  }
}//end for

/* CENTROID VALUE DIVIDES */
//Loop through continuous data elements of centroid
for(int y=0,z=fitPattern.length; y < z; y++)
{
  //Get location of continuous element
  int cLoc = fitPattern[y];

  //If numerical value
  if(isNumeric(centroid[cLoc]))
  {
    //Get numerical value
    double cValue = Double.valueOf(centroid[cLoc]);

    //Divide value by total number of patterns + 1 (for centroid)
```

```
    double newValue = (cValue / (totalPatterns + 1));

    //Round value to 2 decimal places
    newValue = round(newValue, 2, BigDecimal.ROUND_HALF_UP);

    //Convert value back to a string & update centroid value
    centroid[cLoc] = String.valueOf(newValue);
   }
  }//end for
}
else
{
  /* PATTERN VALUE SUMATION */
  //Loop through list of patterns
  for(int i=0,j=totalPatterns; i < j; i++)
  {
   //Get a pattern and split into pieces
   tempPattern = (volume.get(patternList.get(i))).split(delimiter);

   //Loop through pattern elements [Minus labels at end of pattern]
   for(int m=0,n=tempPattern.length-labelDistance; m < n; m++)
   {
    //If numerical value
    if(isNumeric(centroid[m]) && isNumeric(tempPattern[m]))
    {
     //Get numeric value(s)
     double cValue = Double.valueOf(centroid[m]);
     double pValue = Double.valueOf(tempPattern[m]);

     //Sum each NUMERICAL pattern value with respective centroid value
     cValue = cValue + pValue;

     //Convert back to a string & update centroid value
     centroid[m] = String.valueOf(cValue);
    }
   }
  }//end for

  /* CENTROID VALUE DIVIDES */
  //Loop through each centroid value [Minus labels at end of pattern]
  for(int y=0,z=centroid.length-labelDistance; y < z; y++)
  {
   //If numerical value
   if(isNumeric(centroid[y]))
   {
    //Get numerical value
    double cValue = Double.valueOf(centroid[y]);
```

```
        //Divide value by total number of patterns + 1 (for centroid)
        double newValue = (cValue / (totalPatterns + 1));

        //Round value to 2 decimal places
        newValue = round(newValue, 2, BigDecimal.ROUND_HALF_UP);

        //Convert value back to a string & update centroid value
        centroid[y] = String.valueOf(newValue);
       }
     }//end for
   }//end if-else
  }//end run

  private static boolean isNumeric(String str)
  {
    return str.matches("-?\\d+(\\.\\d+)?");  //match a number (latin digits) with optional '-' and decimal.
  }//end isNumeric [Open Source]

  public static double round(double unrounded, int precision, int roundingMode)
  {
    BigDecimal bd = new BigDecimal(unrounded);
    BigDecimal rounded = bd.setScale(precision, roundingMode);
    return rounded.doubleValue();
  }//end round [Open Source]
}//end RunnableCentroid
```