CHAPTER I

INTRODUCTION

Networking forms the intrinsic part of computing. The extensive use of internet makes it prone to vulnerabilities and security threats of new kind coined as “cyber attack”. Globally it made a deficit of $600 billion last year (2017) [2] and this pandemic is expected to cost around $6 trillion annually by 2021[3]. Moreover, India experiences one cybercrime every 10 minutes which is higher than a crime every 12 minutes [4]. Consequently, this necessitated the need for analysing the flow of network traffic beneath the internet. The categorisation of internet traffic based on their application type resist cyber attacks by modelling intrusion prevention system (IPS). Moreover, Peer-to-Peer (P2P) and Voice over IP (VOIP) traffic are being warned of their higher rate of cyber attacks entailing scrutinization [1] [9]. This proposed work formulates a deep learning based network traffic categorization that employs Stacked Auto Encoder (SAE) to gain insight into the encrypted traffic of various applications of P2P and VOIP traffic.

Earlier the port based and payload based techniques were used in segregation of network traffic. Once segregated, they are helpful in network management for monitoring, analysis, prioritization and development of network security devices with stateful inspection. However, the potency of these techniques dwindled with the sprouting of evading techniques that encrypted the traffic. With 70% of attacks expected to use encryption by 2019 [5], machine learning techniques allowed visibility into encrypted traffic based on flow statistics rather than costlier process of analysis by decrypting. The following section briefs the various methods involved in network traffic classification from early stages.

1.1 Port based Classification

Ports are endpoints in communication. They own specific port number which is based on protocol used in communication. It is defined by IANA for well-known services (ex. FTP-20/21, SMTP-25, POP3-110, HTTPS-443). Port based classification categorises the network traffic based on this number. But with the evolution of applications that chose random port numbers (ex. Torrent) and masquerade well-known ports (ex. IM obfuscate the port 80, which is standard port of HTTP) [6], the effectiveness of this technique collapsed.

1.2 Payload based Classification

Payload is the data-part of packets transmitted over network. To overcome the limitations of port based categorization, payload based classification inspects the contents of payload (rather than just port number) to determine the type of traffic. It analyses the various fields of payload to evaluate unique bit patterns in the applications. Distinctive bit patterns of some applications were derived by Thomas (2004) and are tabulated in the following table.

Table 1.1: Patterns of strings commonly found in P2P applications [7]

Protocol/Application	**String Patterns**	**Transport protocol**	**Def. ports**
EDonkey	0xe319010000	TCP/UDP	4661-4665
	0xc53f010000	-	-
BitTorrent	“0x13Bit”	TCP	6881-6889
Gnutella	“GNUT”,”GIV”	TCP	6346-6347
	”GND”	UDP	-

This approach was not successful for all applications because the access to entire fields of payload is restricted due to privacy and technical issues.

1.3 Deep Packet Inspection

Deep Packet Inspection (DPI) mechanisms use signature analysis to understand and verify different applications. Signatures are unique patterns that are associated with every application. In other words, each application is studied for its unique characteristics of signature and a reference database is created. The classification engine then compares the traffic against this reference to identify the exact applications. DPI technique has its own types based on the processing method used for classifying traffic. They include pattern analysis, numerical analysis, behavioural analysis, and heuristic analysis and protocol/state analysis. There are many tools, which are being able to classify the traffic in computer networks. Each of the tools claim to have certain accuracy, but it is a hard task to asses which tool is better, because they are tested on various datasets.

1.3.1 Pattern analysis

Pattern recognition is the automated recognition of patterns and regularities in data. Pattern recognition is closely related to artificial intelligence and machine learning,[1] together with applications such as data mining and knowledge discovery in databases (KDD), and is often used interchangeably with these terms. However, these are distinguished: machine learning is one approach to pattern recognition, while other approaches include hand-crafted (not learned) rules or heuristics; and pattern recognition is one approach to artificial intelligence, while other approaches include symbolic artificial intelligence.

Some applications embed certain patterns (bytes/characters/string) in the payload of the packets, which can be used by the classification engine to identify such protocol. Depending on the application, these patterns may not necessarily be always located at a specific deterministic offset. The patterns might be present in any position in the packet. Still, the classification engine can identify these packets. However, not all protocols embed

special pattern, string or characters in the packets and hence this approach will not work for them.

Pattern recognition algorithms generally aim to provide a reasonable answer for all possible inputs and to perform "most likely" matching of the inputs, taking into account their statistical variation. This is opposed to *pattern matching* algorithms, which look for exact matches in the input with pre-existing patterns. A common example of a pattern-matching algorithm is regular expression matching, which looks for patterns of a given sort in textual data and is included in the search capabilities of many text editors and word processors

1.3.2 Numerical analysis

Numerical analysis involves looking into the numerical characteristics of packets such as payload size, number of response packets, and offsets. Older Skype versions (pre-2.0) are good cases for such analysis. The request from client is an 18-byte message and the response it receives is usually 11 bytes. As the analysis may be spread over multiple packets, the classification decision might take more time.

1.3.3 Behavioural and Heuristic analysis

Occasionally, analysing the traffic behaviour would produce greater insight into the applications that may be running. This behaviour can be used to classify such applications. Similarly, by doing a heuristic analysis of the inspected packets, the underlying protocol can be classified. Behaviour and heuristic analysis typically go hand in hand and many of the antiviral programs use these techniques to identify viruses and worms.

However, all the techniques of DPI required periodical update to keep track with new applications as well as new developments in existing protocols. DPI techniques have also fallen short in classifying the encrypted traffic. As more applications start encrypting traffic, it became a challenge for any classification mechanism to classify the applications accurately. With

encryption, all upper layer information becomes invisible to DPI mechanisms. Advanced behaviour and heuristic analysis methods can help to identify only some applications.

1.4 Statistical Analysis

Statistical approaches exploit application diversity and inherent traffic footprints (flow parameters) to characterize traffic and subsequently derive classification benchmarks through data mining techniques to identify individual applications. This approach overcomes the limitation of port and payload techniques by gaining insight into the network traffic only based on statistical features which are additional payload information. Access to this portion is allowed and moreover these features are independent of encryption mechanisms. Statistical classification is considered light-weight and highly scalable from an operational point of view especially when real-time traffic identification is required. The statistical analysis includes the analysis of the parameters of transmission rate (in packets per second and in bits per second), link load (as fraction of the nominal maximum load), average packet size (in bytes) and graph of the packet size distribution.

1.5 Machine Learning Approaches

Statistical analysis is achieved through machine learning techniques to classify internet traffic accurately and efficiently. Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses in prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning (ML) is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning. Machine learning can also be unsupervised and be used to learn and establish baseline behavioural profiles for various entities and then used to find meaningful anomalies.

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction in commercial use which is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to produce reliable, repeatable decisions and results and uncover hidden insights through learning from historical relationships and trends in the data. The various types of machine learning approaches are detailed below.

1.5.1 Supervised Machine Learning

Supervised learning is based on attributes of a class i.e. samples are chosen on the basis of attributes collected by the whole data. The input data is provided with a collection of sample instances, pre-classified into classes. The output of the learning process is a classification model that is constructed by examining generalization from providing instances. Classification approaches mainly have two phases (steps), training and testing. Learning phase that examine the provided data (called the training dataset) and constructs (builds) a classification model. And the model that has been built in the training phase is used to classify new unseen instances. This is similar to the fact as if the computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. Supervised learning is where one may have input variables (x) and an output variable (Y). To learn the mapping function from the input to the output is given in eqn. (1)

$$Y = f(x) \qquad \text{eqn. (1)}$$

The goal is to approximate the mapping function so well that when one have new input data (x) that one can predict the output variables (Y) for that data. It is called supervised learning because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. Here one may know the correct answers. The algorithm iteratively makes

predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

1.5.2 Unsupervised Machine Learning

Unsupervised learning techniques use the concept where there is no need of the training phase. No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning). Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from unlabelled data. Unsupervised learning is has input data (x) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning there are no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

1.5.3 Semi-Supervised Machine Learning

Semi-supervised learning is a class of supervised learning tasks and techniques that also make use of unlabelled data for training – typically a small amount of labelled data with a large amount of unlabelled data. Semi-supervised learning falls between unsupervised learning (without any labelled training data) and supervised learning (with completely labelled training data). Many machine-learning researchers have found that unlabelled data, when used in conjunction with a small amount of labelled data, can produce considerable improvement in learning accuracy. Problems where one have a large amount of input data (X) and only some of the data is labelled (Y) are called semi-supervised learning problems. These problems lies in between both supervised and unsupervised learning. A good example is a photo archive where only some

of the images are labelled, (e.g. dog, cat, person) and the majority are unlabelled. Many realworld machine learning problems fall into this area. This is because it can be expensive or time-consuming to label data as it may require access to domain experts. Whereas unlabelled data is cheap and easy to collect and store. One can use unsupervised learning techniques to discover and learn the structure in the input variables. One can also use unsupervised learning techniques to make best guess predictions for the unlabelled data, feed that data back into the supervised learning algorithm as training data and use the model to make predictions on new unseen data.

Various algorithms are used in the implementation of machine learning approaches. Some of the advanced algorithms in ML employed in network traffic classification are discussed in section 1.6.

1.6 Application of ML approaches in internet traffic classification:

1.6.1 Bayes Net

Bayes net approach generally known as Belief Network is a probabilistic model which uses the graph model to represent the set of random variables and their conditional dependencies. Bayes net uses the concept of directed acyclic graph (DAG) to represent the set, in which each node represent a variable and edges among the nodes represent the relative dependencies between random variables and these relative dependencies in the graph are calculated by well known statistical and computational methods. This Bayes net gives good classification accuracy with full feature dataset.

1.6.2 Feed forward Neural Network

The feed forward Neural Network (NN) was the first and simplest type of artificial NN methods. A feed forward neural network is an artificial neural network wherein connections between the units do not form a cycle. As such, it

is different from recurrent neural networks. In this network, the information moves in forward direction (i.e) from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. It gives high classification accuracy for smaller datasets.

1.6.3 C4.5 Decision tree

C4.5 is a popular decision tree ML algorithm used to develop Univariate decision tree. C4.5 is an enhancement of Iterative Dichotomiser 3 algorithm which is used to find simple decision trees. C4.5 is also called a statistical classifier because of its good ability of classification. C4.5 makes decision trees from a set of training data samples, with the help of information entropy concept. The training data set contains of a greater number of training samples which are characterized by different attributes and it also consists of the target class. C4.5 selects a particular attribute of the data at each node of the tree which is used to split its set of data samples into subsets in one or another class. It is based on the criterion of normalized information gain that is obtained by selecting an attribute for splitting the data. The attribute with the highest normalized information gain is chosen and made a decision. After that, the C4.5 algorithm repeats the same action on the smaller subsets. C4.5 has made various improvements to ID3, like it can handle both continuous attributes and discrete attributes, it can handle training data with missing attribute values, it can also handle attributes with differing costs etc resulting with higher accuracy for large datasets.

1.6.4 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a data clustering algorithm. It is a density based clustering algorithm; it finds the number of clusters starting from the estimated density distribution of

the corresponding nodes. It yields good accuracy in classification but it is not entirely deterministic. It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature

1.6.5 K-Means

K-Means clustering algorithm is a partitioned-based algorithm; it partitioned objects of a dataset into K disjoint subsets. It maximizes the homogeneity of the cluster and minimizes the square-error, where square-error calculated as the distance between each object and the center or mean of a cluster. The centers of K cluster are initially chosen randomly and after that dataset partitioned into nearest cluster. K-Means iteratively computes new centres and clusters respectively. This process continues until the clusters are stabilized.

The algorithm has a loose relationship to the *k*-nearest neighbour classifier, a popular machine learning technique for classification that is often confused with *k*-means due to the name. Applying the 1-nearest neighbour classifier to the cluster center's obtained by *k*-means classifies new data into the existing clusters. This is known as nearest centroid classifier .

1.6.6 Expectation Maximization

Expectation Maximization (EM) is an iterative method for looking maximization likelihood parameters and produce clusters. There are mainly two steps in EM method. First is expectation step and the second one is maximization step. In first step, estimate what parameter is using random numbers and the second step which uses mean and variance to re-estimate the

parameter. This process continuously proceeds until then they reach with a local maximization.

1.6.7 C5.0 Decision Tree

The C5.0 decision tree classifier is faster being multithreaded than C4.5 classifier. It works with more improved accuracy, speed and memory storage than other decision tree classifiers. C5.0 provides the option automatically that winnows the attributes to remove those that may be unhelpful. The C5.0 rule sets have lower error rates on unseen cases. C5.0 algorithm gives the acknowledgement on noise and missing data. Problem of over fitting and error pruning is solved by the C5.0 algorithm. In classification technique the C5.0 classifier can anticipate which attributes are relevant and which are not relevant in classification. It is easy to implement, it can deal with both values categorical and continuous values and noise.

1.6.8 Stacked Auto Encoder

Autoencoder is a kind of unsupervised learning structure that owns three layers: input layer, hidden layer, and output layer. The process of autoencoder training consists of two parts: encoder and decoder. Encoder is used for mapping the input data into hidden representation, and decoder is referred to reconstructing input data from the hidden representation. The structure of Stacked Auto Encoder (SAE) is stacking n autoencoder into n hidden layers by an unsupervised layer wise learning algorithm and then fine-tuned by a supervised method. So the SAEs based method can be divided into three steps:

(1) Train the first autoencoder by input data and obtain the learned feature vector;

(2) The feature vector of the former layer is used as the input for the next layer, and this procedure is repeated until the training completes.

(3) After all the hidden layers are trained, Back Propagation algorithm (BP) is used to minimize the cost function and update the weights with labelled training set to achieve fine tuning.

1.7 Introduction to Deep Learning based network traffic classification

Deep learning (DL) is a ML technique inspired by the working of human brain. At each level it learns to transform its input data into a slightly more abstract and composite representation. The salient nature of this technique is that it can learn which features to optimally place in which level on its own. In this proposed work, the network traffic of various applications of P2P and VOIP are extracted for the additional payload information using wireshark network analyser. They are pre-processed by handling missing values and fed into SAE. The autoencoders are trained in an unsupervised, greedy, layer-wise fashion and learns about the weights which are used to minimise reconstruction error [8]. This in turn acts as good starting point to initialize a network for classification/similarity. It is then finely tuned by regression layer added for supervised learning with labelled dataset. This yields the discriminative features that assist in categorizing the network traffic based on statistics.

CHAPTER II
LITERATURE SURVEY

TITLE: Early Application Identification.

AUTHORS: Laurent Bernaille, Renata Teixeira et al

PUBLICATION: CONEXT'06, ACM 2006

This method deals with early identification of application. As systematic analysis is slow and flow statistics is limited to offline classification, this approach performs classification based on inspection of first few packets. It distinguishes the behaviour of an application from size and direction of first P packets of the TCP connection. Parameters of mean packet size and variance with direction of traffic are used in distinguishing application in GMM clustering. The GMM clustering combined with TCP port numbers classifies 98% of known applications correctly. It can label new applications as unknown or masquerade. But it faces limitations as it requires first four packets of TCP connection to be in correct order. If packet sampling is used, method degrades and is subject to evasion.

Inference: To avoid evasion, classifier modelling requires both offline and online phases in modelling of classifier.

TITLE: Offline/Real-time Traffic classification using Semi-Supervised Learning

AUTHORS: Jeffrey Erman, Anirban Mahanti et al

PUBLICATION: Performance Evaluation**,** Elsevier 2007

This work outlooks offline and real time traffic classification of packets accommodating both known and unknown flows. It strived for longevity of classifiers and need for retraining of classifiers. They considered parameters of flow statistics addressing high byte and flow accuracy. Here fully labelled training data is eliminated by using semi-supervised technique consisting of few

labelled and many unlabelled flow statistics without payload signatures and patterns. It was determined that flow statistics creates longevity up to high accuracy of 95%. Semi-supervised technique handles both known and unknown flows and detects retraining efficiently. The retraining point is detected but further processing techniques are only suggested. It faced issues with the mice and elephant flows of traffic which was a complication to the design of classifier. Retraining is left to network operators for enhancing the classifier's performance.

Inference: Combination of supervised and unsupervised techniques improves efficiency.

TITLE: A survey of techniques for internet traffic classification in machine learning.

AUTHORS: Thuy T.T.Nguyen, Grenville Armitage

PUBLICATION: IEEE Communications Surveys & Tutorials, 2008.

It surveys various Machine Learning (ML) techniques used in internet traffic classification. The different ML algorithms for offline and online analysis are studied. Requirements for ML based real-time classification are outlined. The algorithms Auto Class, EM, Decision Tree and Naive Bayes demonstrated high accuracy. It suggested the need for parallel processing of algorithms to cope with millions of concurrent flows.

Inference: Parallel processing of flows can be tackled effectively by deep learning networks.

TITLE: Early Classification of Network Traffic through Multi-Classification.

AUTHORS: Alberto Dainotti, Antonio Pescape et al

PUBLICATION: Traffic monitoring and analysis, Springer 2011

This work deals with multi-classification of network traffic. It employs combination of algorithms for traffic classification. It manifests that accuracy of

standalone algorithms can be enhanced significantly by advanced combination strategy of machine learning and pattern recognition. It used features from limited number of packets. Selecting complementary classifiers of combination algorithms allow further improvement in terms of accuracy. Yet it had computational complexity and time-related issues.

Inference**:** Repetitive combination of training improves efficiency. This can be done with less complexity in deep learning.

TITLE: Session level flow classification by packet size distribution and grouping

AUTHORS: Chun-Nan Lu, Chun-Ying Huang et al

PUBLICATION: Computer Networks, Elsevier 2011

This work performs session level flow classification that classifies network traffic based on behaviours observed in applications. It performs session based analysis instead of flow level. It classifies without examining packet payloads by segmenting the network flows as session to identify the application. Parameters considered include packet size distribution and port locality. It achieved high accuracy rates in both flow and session classification with low error rates. However the underlying traffic within the flow is also segregated as application flow.

Inference**:** Packet Size distribution is a potential parameter to be considered in classification.

TITLE: An Effective Network Traffic Classification Method with Unknown flow detection.

AUTHORS: Jun Zhang, Chao Chen et al

PUBLICATION: IEEE Transactions on network & service management, IEEE 2013

This work considered the issue of classification performance which is severely affected by limited supervised information and unknown applications. The parameters of flow statistics and correlation information are utilized. Correlation information with statistical features is used to detect unknown flows. The proposed work was compared with existing methods of KNN, NB, C4.5. But it encountered complications for flows captured at different time on network

Inference:

The features considered by the classifier must be independent of the time at which the data is captured.

TITLE: Internet Traffic classification by Aggregating Correlated Naïve Bayes Prediction.

AUTHORS: Jun Zhang, Chao Chen et al.

PUBLICATION: IEEE transactions of information forensics and security,2013.

The objective of this work is to improve performance when few training data is available. It evaluated that traffic classification can be enhanced by feature discretization. Naïve Bayes with feature discretization and reduction gives high accuracy and faster classification. Effective feature discretization dramatically affects the performance.

Inference**:** Feature reduction improves potent of classifier. Our proposed work combines feature reduction with classifier as hidden layer compact the features in deep learning.

TITLE: Internet traffic classification based on flows' statistical properties.

AUTHORS: Alina Vladutu, Dragos Comaneci et al

PUBLICATION: International Journal of Network Management, Wiley Library 2016

This proposal performed classification based on statistical parameters with machine learning technique. It classified the network traffic with flows' parameters. Unidirectional and Bidirectional flows of network traffic are considered as parameters. It analyzed the statistical features and designed a classifier to detect new flows. It resulted in 90% of flows classified accurately as unknown.

Inference**:** Additional payload information (flow parameters) is light-weight and helpful in distinguishing the encrypted network traffic.

TITLE: A Comparative performance analysis on network traffic classification using supervised learning algorithms.

AUTHORS: Archanaa, Athulya et al

PUBLICATION: ICACCS,IEEE 2017.

This work compares the performance of advanced supervised algorithms in network traffic classification. Reliable features are selected by wrapper and filter depicting the importance of feature reduction. It demonstrated that feature reduction reduces computational time to train the system and produce better results. Bayes Net performed better than Naive Bayes, Complement naive Bayes algorithms. Among ensemble classifiers, DECORATE; an ensemble classifier out-performs state-of-art ensemble classifiers. Out of rule based classifier JRip, oneR shows high accuracy.

Inference: Deep learning encloses feature reduction with dropout factors while classifying. This further reduces computational complexity than ensemble classifiers.

TITLE: Application Identification via Network Traffic Classification.

AUTHORS: Baris Yamansavascilar et al

PUBLICATION: CNC,2017,IEEE

This work outlines application identification by classifying the traffic. It analyzed more contributing features to distinguish the application. UNB ISCX data is combined with internal dataset and is used in training. It evaluated the four classification algorithms, namely J48, Random Forest, k-NN, and Bayes Net. With the complete set of 111 features, k-NN gave the best result for the ISCX Dataset as 93.94% of accuracy and Random Forest gave the best result for the internal dataset as 90.87% of accuracy

Inference: A combination of dataset increases the robustness of classifier. Feature reduction without comprise of accuracy improves computational performance and efficiency.

TITLE: Multi-Classification approaches for classifying mobile app traffic.
AUTHORS: Giuseppe, Domenico et al
PUBLICATION: Preprint to Journal of Network and Computer Applications, 2017, Elsevier.

This work employs hard and soft combiners of classifiers like Naive Bayes, Multinomial NB, Random Forest, Support Vector Classifier, Decision Tree. The parameters considered include packet length, minimum, maximum, mean, median, variance, absolute deviation and standard deviation. Network traffic data was considered from mobile application. The fusion of classifiers resulted in high performance with removal of zero-payload packets.

Inference:

Future directions were suggested for deeper analysis of classifiers with specifically optimized set of features.

Thus based on the survey, a new approach for network traffic classification must be capable of handling feature reduction to enhance accuracy and deeper analysis of statistical parameters including packet size distribution. Additionally, it must be computationally light and must be independent of

packets collected over different time in network ensuring longevity. This can be implemented with Stacked Auto Encoders which are light and are capable of reducing feature set as an integrated part of training.

CHAPTER III

METHODOLOGY

3.1PROPOSED SYSTEM

The need for categorizing the ground-truth network traffic even if it is encrypted and obfuscated by Virtual Private Network (VPN) or other tunnelling mechanisms acted as the motivation of this work. The proposed work aims to categorise the network traffic of P2P and VOIP traffic of five applications namely eDonkey, Quake 3, BitTorrent, Skype, Facebook. The overview of the proposed work is depicted in Figure3.1.

3.1 Modules of the system

The phases involved in the proposed system are as follows.

(i) Data Extraction.

(ii) Data Pre-processing.

(iii) Training the Stacked Auto Encoder.

(iv) Evaluation of classifier.

3.2.1 Data Extraction

Network packet consists of both header and payload portion. Header contains information about length of packet, packet number, protocol, destination and source address. For analysis this header section including statistics is extracted. From the entire network packet, the statistical parameters to be processed are extracting by wireshark network analyser. It can done using statistics option available in it. The conversations from the pcap file is extracted as .csv for further analysis.

3.2.2 Data Pre-processing

The extracted data is analysed for handling missing or incomplete data. The missing values are replaced with null values. The corrupted values and unavailable values are also replaced with null values. Packet length is an important parameter to be considered. But it cannot be directly extracted with

the wireshark analyser. Hence it is extracted individually from those packets and added to extracted data. This pre-processed data is fed into the classifier for training.

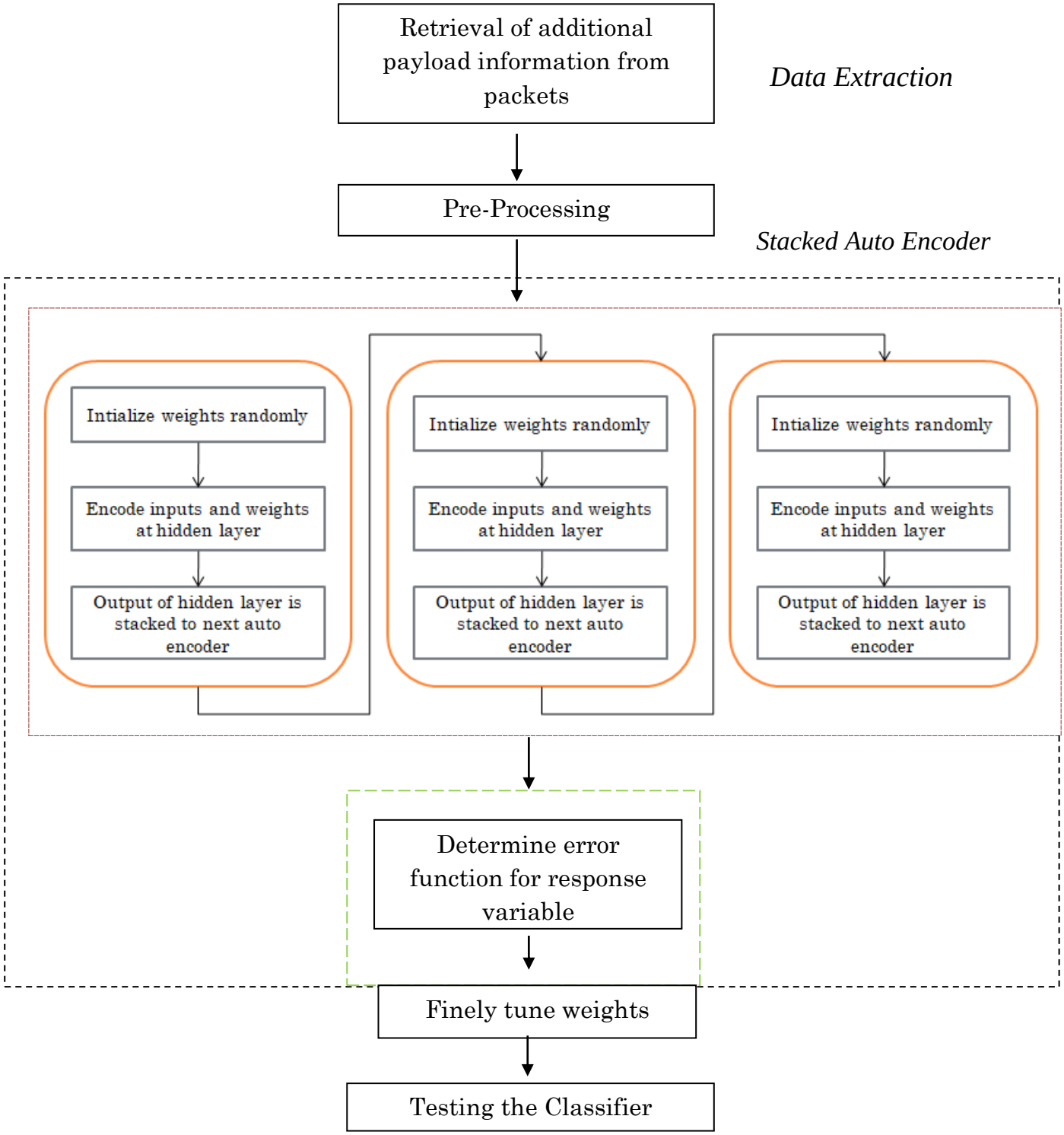


Fig 3.1 System Architecture

3.2.3 Training Stacked Auto Encoder

Pre-processed data of csv file is fed into SAE for training. Weights are initialized randomly and intaken by the deep learner SAE. The general working of SAE is depicted in Figure 3.2. The data is analysed at each level for more compact feature that can uniquely identify the application. SAE initally begins with unsupervised learning examining the input data for distinguishing features . These features are fed into hidden layer for further condensation to determine the effective features. Finally the supervised learning is done at softmax layer that fines tunes the features for design of precise classifier.

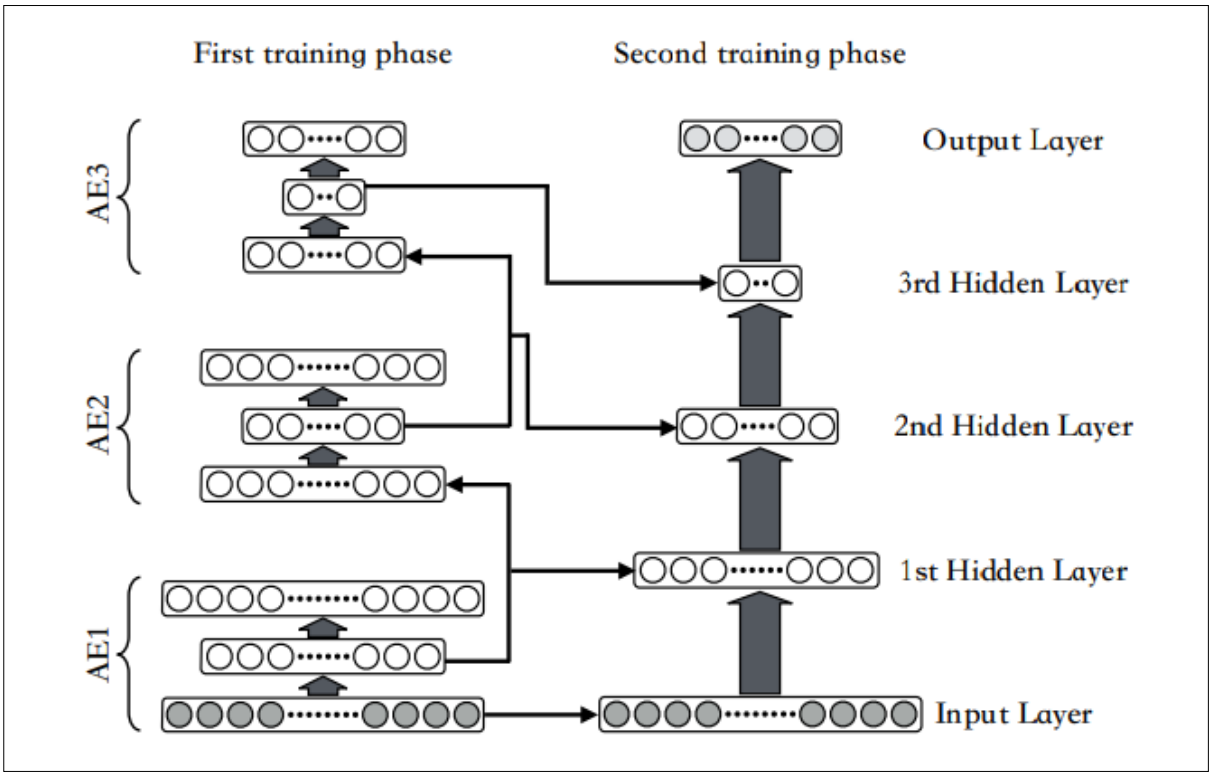


Fig 3.2 Structure of SAE

3.2.4 Evaluation of classifier

Once the classifier is trained or modelled, it is tested with the test data. The test data is also processed by extraction and pre-processing and sentenced to testing. Based on the results, the accuracy and performance of classifier is

evaluated. It is determined from the misclassification table with true positive and false positive rate.

3.3 Stacked Auto Encoder

Neural networks aim to find a non-linear relationship between input *x* with output *y*, as *y=f(x)***.** An autoencoder is a form of unsupervised neural network which tries to find a relationship between features in space such that *h=f(x)*, which helps to learn the relationship between input space and can be used for data compression, dimensionality reduction, and feature learning. An autoencoder consists of an encoder and decoder. The encoder helps encode the input *x* in a latent representation *y*, whereas a decoder converts back the *y* to *x*. Both the encoder and decoder possess a similar representation of form. The stacked autoencoder is an approach to train deep networks consisting of multiple layers trained using the greedy approach. Figure3.3 demonstrates a stacked autoencoder with two layers. A stacked autoencoder can have *n* layers, where each layer is trained using one layer at a time called ladder-wise pre-training. Initially the system condenses the features greedily in unsupervised manner and finally refines them in supervised manner. The SAE in Figure3.3 will be trained as in Figure3.4.

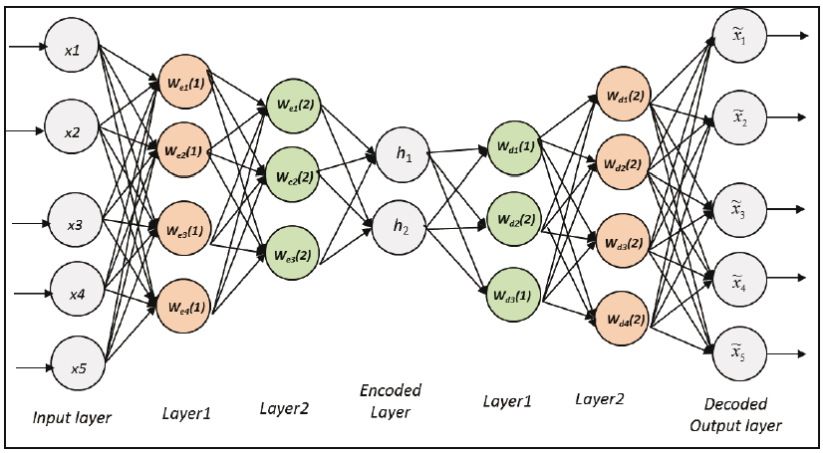


Fig3.3 SAE with two hidden layers

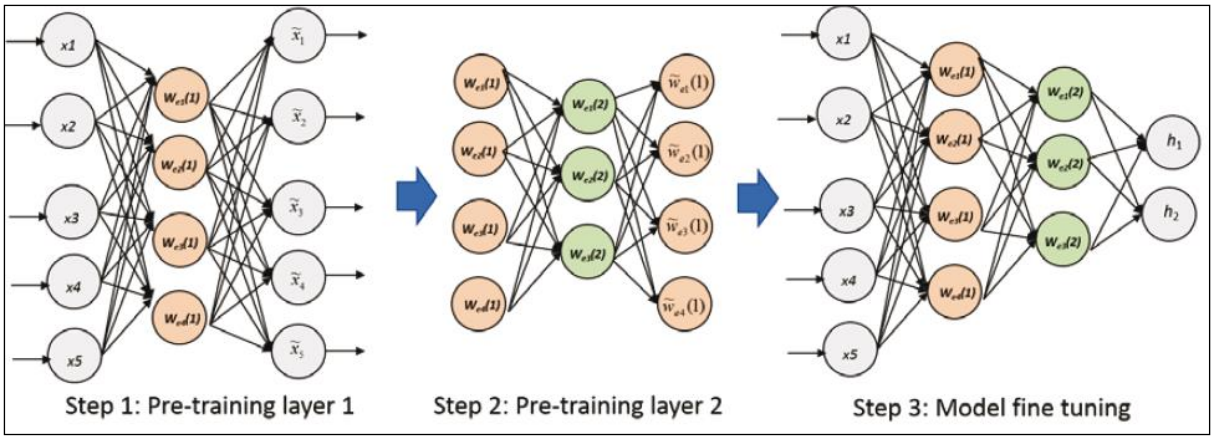


Fig 3.4 Training of SAE in Fig3.3

The initial pre-training of layer 1 is obtained by training it over the actual input x_i . The first step is to optimize the *W(1)* layer of the encoder with respect to output X. The second step in the preceding example is to optimize the weights *W(2)* in the second layer, using *W(1)*as input and output. Once all the layers of *W(i)* where *i*=1, 2, ...,*n* is number of layers are pre-trained, model fine-tuning is performed by connecting all the layers together.

Input: Set of input features *x* labelled with class *y*

Output: Trained model with discriminative features of *x*

Pseudocode:

1 $x \leftarrow x_1, x_2, \ldots, x_n$ *// set of inputs*

2 **for each** hidden layer (h_i)

3 $h_i \leftarrow x.w_i$

4 $\tilde{x} \leftarrow h_i.\tilde{w}_I$ *// $\tilde{x}$ is compact features that represent x*

5 **calculate** E = $\| x - \tilde{x} \|$ *// Euclidean norm*

6 **update** $w_i, \tilde{w}_i$ **until** *E=0*

// To stack encoders

7 $x \leftarrow z_i$ *//output of h_i ($z_i \leftarrow f_{nl}(w_i.[x,i]^T)$)*

8 **end for**

9 $\tilde{y} \leftarrow x.w_{n-1}$

10 **calculate** E = $\| y - \tilde{y} \|$

11 **update All** $w_i, \tilde{w}_i$ **until** *E=0* *// Fine-tuning*

SYSTEM REQUIREMENTS

4.1 Hardware Requirements:

Processor	Intel® Pentium® CPU A1018 @ 2.10GHZ
RAM	2.00 GB(1.89 GB USABLE)
Hard Disk	232 GB
Monitor	15 Inch colour monitor
Keyboard	Standard Keyboard

4.2 Software Requirements:

Operating System	Ubuntu 16.04
Tool	RStudio, Wireshark
Back End	Microsoft Excel

CHAPTER V

RESULTS AND DISCUSSION

5.1 Experimental Setup

The work was done experimentally with Dell personal computer running Windows7 Ultimate OS and Intel core i3 of capacity 4GB RAM. Implementation was done with R language in R studio. Network packets of five applications (P2P and VOIP) were collected from the benchmark dataset of CAIDA and UNB ISCX VPN-non VPN dataset. They are available in .pcap format and capable of being processed by wireshark network analyser.

5.2 Implementation

5.2.1 Data Extraction

Wireshark includes filters, colour-coding and other features that help to dig deep into network traffic and inspect individual packets. Wireshark is also capable of capturing packets that is transmitted to or from the system that is used in capturing the traffic. Fig 4.1(a) and (b) depicts the wireshark tool analysing Skype and Facebook .pcap files.

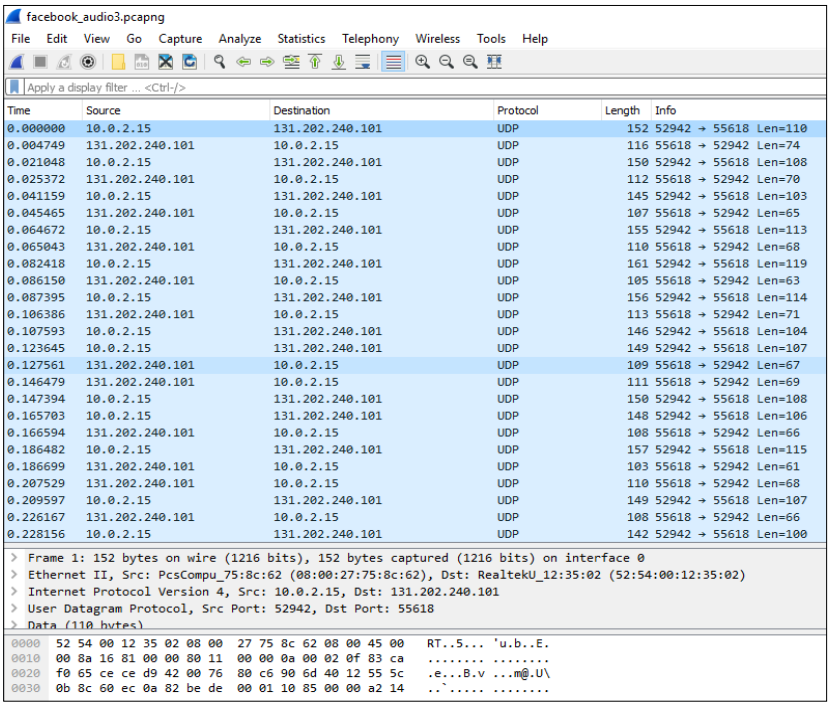


Fig 5.1(a) Facebook .pcap file analysed in wireshark

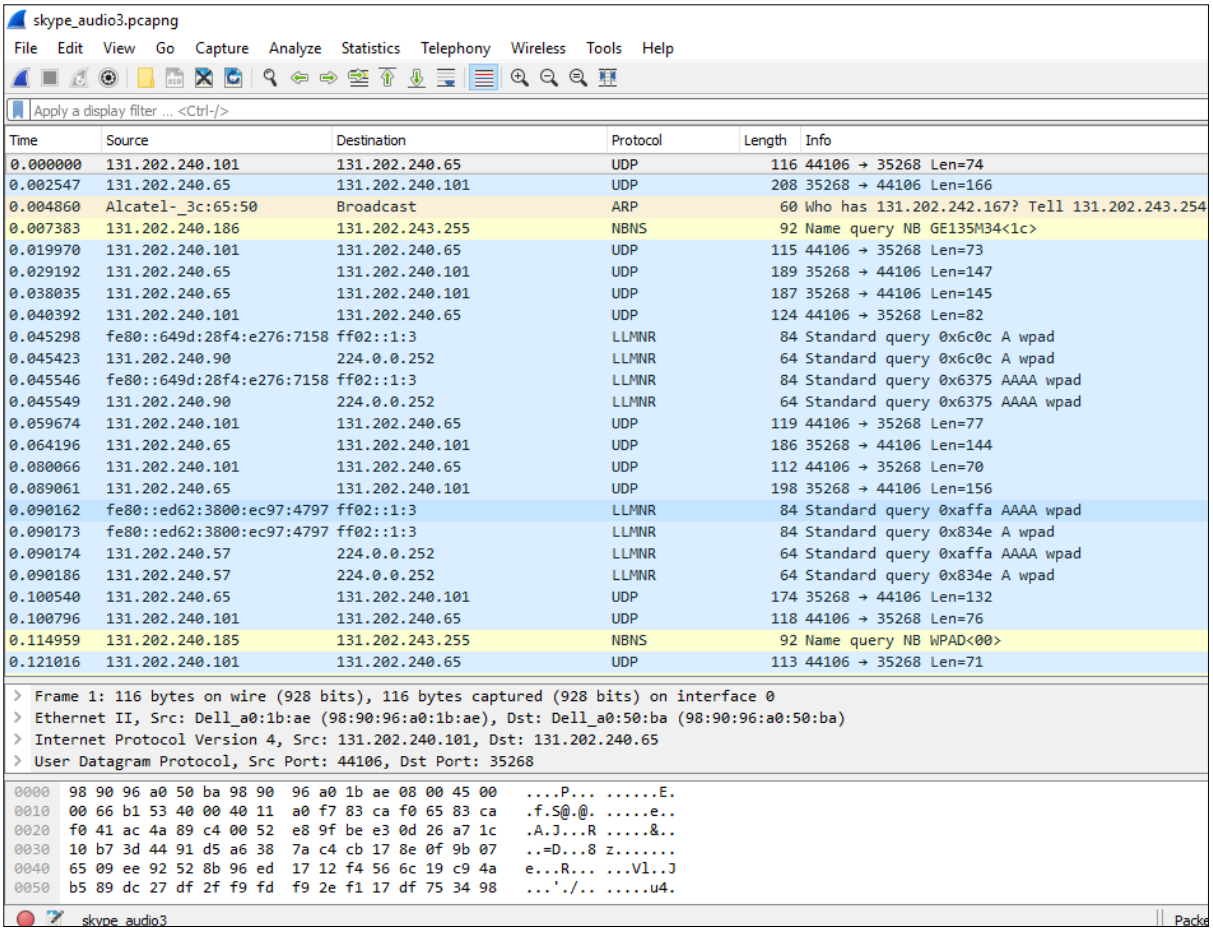


Fig 5.1(b) Skype .pcap file analysed in wireshark

The packets are then extracted with the accessible, light-weight header information. Statistical parameters are extracted by Statistics->Conversations option available in wireshark. It is then converted to .csv file for importing data in training. It is depicted in Figure 4.2.

5.2.2 Pre-processing

The data extracted from the packets is further processed for handling missing values. It is replaced with null values. Packet size being an important parameter is included manually by exporting the basic information of file in .csv format. Once it is pre-processed, it is made ready for importing the data. The .csv file of Skype and Facebook at the end of pre-processing phase is shown below in Fig 4.3 (a) and (b).

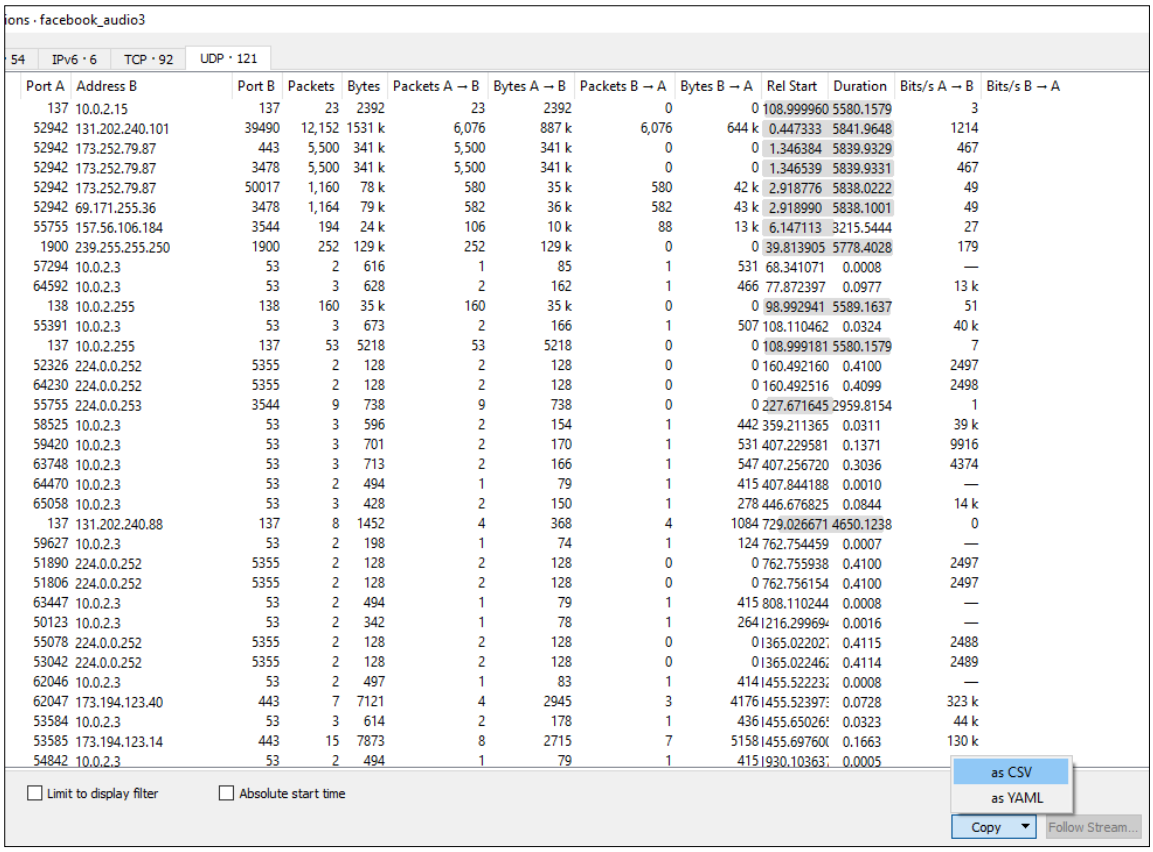
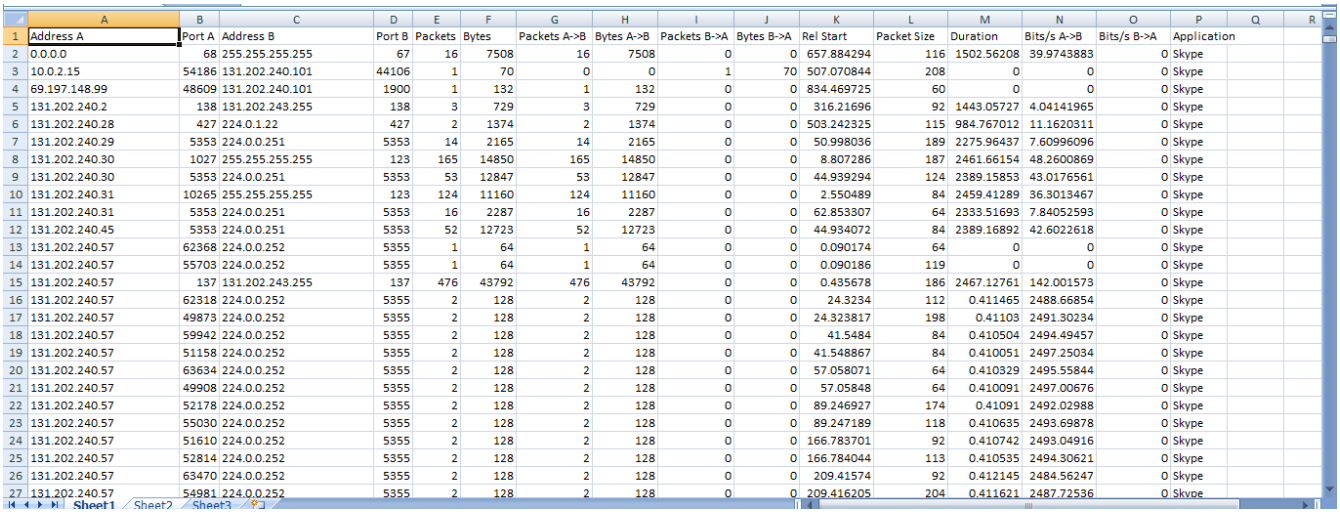


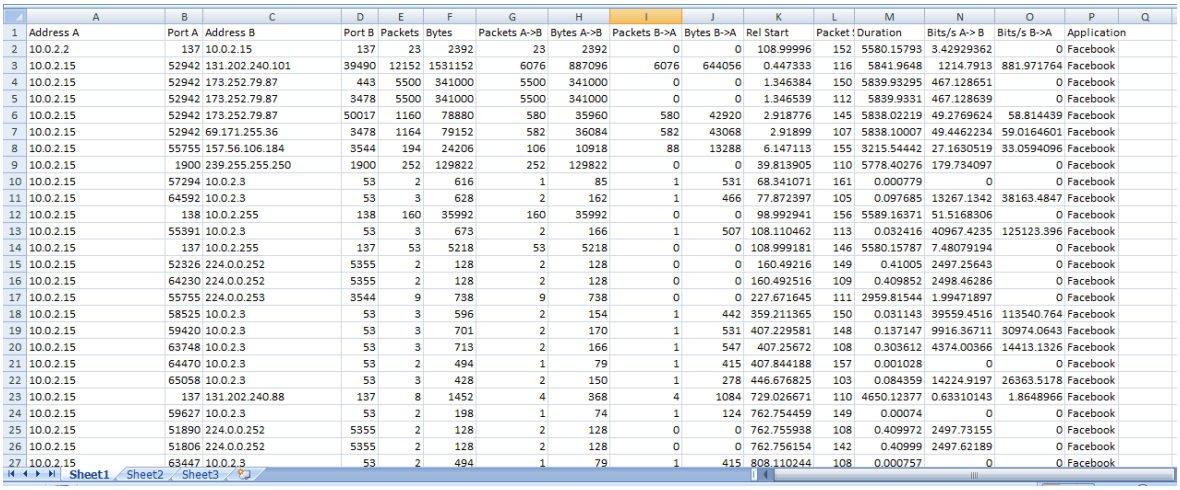
Fig 5.2 Extracting statistical parameters from .pcap file



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets B->A	Bytes B->A	Rel Start	Packet Size	Duration	Bits/s A->B	Bits/s B->A	Application
2	0.0.0.0	68	255.255.255.255	67	16	7508	16	7508	0	0	657.884294	116	1502.56208	39.9743883	0	Skype
3	10.0.2.15	54186	131.202.240.101	44106	1	70	0	0	1	70	507.070844	208	0	0	0	Skype
4	69.197.148.99	48609	131.202.240.101	1900	1	132	1	132	0	0	834.469725	60	0	0	0	Skype
5	131.202.240.2	138	131.202.243.255	138	3	729	3	729	0	0	316.21696	92	1443.05727	4.04141965	0	Skype
6	131.202.240.28	427	224.0.1.22	427	2	1374	2	1374	0	0	503.242325	115	984.767012	11.1620311	0	Skype
7	131.202.240.29	5353	224.0.0.251	5353	14	2165	14	2165	0	0	50.998036	189	2275.96437	7.60996096	0	Skype
8	131.202.240.30	1027	255.255.255.255	123	165	14850	165	14850	0	0	8.807286	187	2461.66154	48.2600869	0	Skype
9	131.202.240.30	5353	224.0.0.251	5353	53	12847	53	12847	0	0	44.939294	124	2389.15853	43.0176561	0	Skype
10	131.202.240.31	10265	255.255.255.255	123	124	11160	124	11160	0	0	2.550489	84	2459.41289	36.3013467	0	Skype
11	131.202.240.31	5353	224.0.0.251	5353	16	2287	16	2287	0	0	62.853307	64	2333.51693	7.84052593	0	Skype
12	131.202.240.45	5353	224.0.0.251	5353	52	12723	52	12723	0	0	44.934072	84	2389.16892	42.6022618	0	Skype
13	131.202.240.57	62368	224.0.0.252	5355	1	64	1	64	0	0	0.090174	64	0	0	0	Skype
14	131.202.240.57	55703	224.0.0.252	5355	1	64	1	64	0	0	0.090186	119	0	0	0	Skype
15	131.202.240.57	137	131.202.243.255	137	476	43792	476	43792	0	0	0.435678	186	2467.12761	142.001573	0	Skype
16	131.202.240.57	62318	224.0.0.252	5355	2	128	2	128	0	0	24.3234	112	0.411465	2488.66854	0	Skype
17	131.202.240.57	49873	224.0.0.252	5355	2	128	2	128	0	0	24.323817	198	0.41103	2491.30234	0	Skype
18	131.202.240.57	59942	224.0.0.252	5355	2	128	2	128	0	0	41.5484	84	0.410504	2494.49457	0	Skype
19	131.202.240.57	51158	224.0.0.252	5355	2	128	2	128	0	0	41.548867	84	0.410051	2497.25034	0	Skype
20	131.202.240.57	63634	224.0.0.252	5355	2	128	2	128	0	0	57.058071	64	0.410329	2495.55844	0	Skype
21	131.202.240.57	49908	224.0.0.252	5355	2	128	2	128	0	0	57.05848	64	0.410091	2497.00676	0	Skype
22	131.202.240.57	52178	224.0.0.252	5355	2	128	2	128	0	0	89.246927	174	0.41091	2492.02988	0	Skype
23	131.202.240.57	55030	224.0.0.252	5355	2	128	2	128	0	0	89.247189	118	0.410635	2493.69878	0	Skype
24	131.202.240.57	51610	224.0.0.252	5355	2	128	2	128	0	0	166.783701	92	0.410742	2493.04916	0	Skype
25	131.202.240.57	52814	224.0.0.252	5355	2	128	2	128	0	0	166.784044	113	0.410535	2494.30621	0	Skype
26	131.202.240.57	63470	224.0.0.252	5355	2	128	2	128	0	0	209.41574	92	0.412145	2484.56247	0	Skype
27	131.202.240.57	54981	224.0.0.252	5355	2	128	2	128	0	0	209.416205	204	0.411621	2487.72536	0	Skype

Sheet1 Sheet2 Sheet3

Fig 5.3 (a) Pre-processed Skype .csv file



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets B->A	Bytes B->A	Rel Start	Packet	Duration	Bits/s A-> B	Bits/s B->A	Application	
2	10.0.2.2	137	10.0.2.15	137	23	2392	23	2392	0	0	108.99996	152	5580.15793	3.42929362	0	Facebook	
3	10.0.2.15	52942	131.202.240.101	39490	12152	1531152	6076	887096	6076	644056	0.447333	116	5841.9648	1214.7913	881.971764	Facebook	
4	10.0.2.15	52942	173.252.79.87	443	5500	341000	5500	341000	0	0	1.346384	150	5839.93295	467.128651	0	Facebook	
5	10.0.2.15	52942	173.252.79.87	3478	5500	341000	5500	341000	0	0	1.346539	112	5839.9331	467.128639	0	Facebook	
6	10.0.2.15	52942	173.252.79.87	50017	1160	78880	580	35960	580	42920	2.918776	145	5838.02219	49.2769624	58.814439	Facebook	
7	10.0.2.15	52942	69.171.255.36	3478	1164	79152	582	36084	582	43068	2.91899	107	5838.10007	49.4462234	59.0164601	Facebook	
8	10.0.2.15	55755	157.56.106.184	3544	194	24206	106	10918	88	13288	6.147113	155	3215.54442	27.1630519	33.0594096	Facebook	
9	10.0.2.15	1900	239.255.255.250	1900	252	129822	252	129822	0	0	39.813905	110	5778.40276	179.734097	0	Facebook	
10	10.0.2.15	57294	10.0.2.3	53	2	616	1	85	1	531	68.341071	161	0.000779	0	0	Facebook	
11	10.0.2.15	64592	10.0.2.3	53	3	628	2	162	1	466	77.872397	105	0.097685	13267.1342	38163.4847	Facebook	
12	10.0.2.15	138	10.0.2.255	138	160	35992	160	35992	0	0	98.992941	156	5589.16371	51.5168306	0	Facebook	
13	10.0.2.15	55391	10.0.2.3	53	3	673	2	166	1	507	108.110462	113	0.032416	40967.4235	125123.396	Facebook	
14	10.0.2.15	137	10.0.2.255	137	53	5218	53	5218	0	0	108.999181	146	5580.15787	7.48079194	0	Facebook	
15	10.0.2.15	52326	224.0.0.252	5355	2	128	2	128	0	0	160.49216	149	0.41005	2497.25643	0	Facebook	
16	10.0.2.15	64230	224.0.0.252	5355	2	128	2	128	0	0	160.492516	109	0.409852	2498.46286	0	Facebook	
17	10.0.2.15	55755	224.0.0.253	3544	9	738	9	738	0	0	227.671645	111	2959.81544	1.99471897	0	Facebook	
18	10.0.2.15	58525	10.0.2.3	53	3	596	2	154	1	442	359.211365	150	0.031143	39559.4516	113540.764	Facebook	
19	10.0.2.15	59420	10.0.2.3	53	3	701	2	170	1	531	407.229581	148	0.137147	9916.36711	30974.0643	Facebook	
20	10.0.2.15	63748	10.0.2.3	53	3	713	2	166	1	547	407.25672	108	0.303612	4374.00366	14413.1326	Facebook	
21	10.0.2.15	64470	10.0.2.3	53	2	494	1	79	1	415	407.844188	157	0.001028	0	0	Facebook	
22	10.0.2.15	65058	10.0.2.3	53	3	428	2	150	1	278	446.676825	103	0.084359	14224.9197	26363.5178	Facebook	
23	10.0.2.15	137	131.202.240.88	137	8	1452	4	368	4	1084	729.026671	110	4650.12377	0.63310143	1.8648966	Facebook	
24	10.0.2.15	59627	10.0.2.3	53	2	198	1	74	1	124	762.754459	149	0.00074	0	0	Facebook	
25	10.0.2.15	51890	224.0.0.252	5355	2	128	2	128	0	0	762.755938	108	0.409972	2497.73155	0	Facebook	
26	10.0.2.15	51806	224.0.0.252	5355	2	128	2	128	0	0	762.756154	142	0.40999	2497.62189	0	Facebook	
27	10.0.2.15	63447	10.0.2.3	53	2	494	1	79	1	415	808.110244	108	0.000757	0	0	Facebook	

Sheet1 Sheet2 Sheet3

Fig 5.3 (b) Pre-processed Facebook .csv file

5.2.3 Training the SAE

The pre-processed .csv file is fed into Stacked Auto Encoder with three hidden layers. It initially learns the data to gain insight by unsupervised autoencoder and finally tuned with supervised learning with regard to the response variable. The implementation is done using H2O package available in R. H2O is a very popular open source library that is used to build machine learning models. It is produced by H2O.ai and supports multiple languages including R and Python. The H2O package is a multipurpose machine learning library developed for a distributed environment to run algorithms on big data. *Tanh* is the activation function deployed, which is a rescaled and shifted logistic function and it allows the training algorithm to converge faster. The training of classifier begins with initialization of thread in H2O as in Fig 4.4. Then the autoencoder are loaded with input data and based on weights, the hidden layer compacts the features and refines the weight. Finally the response variable tunes the model to optimise the weights to global optimum. The model built by the classifier for future evaluation is depicted in Fig 4.5.

5.2.4 Evaluation of Classifier

Based on the classifier model constructed, test data of CAIDA and UNB ISCX is used to evaluate the accuracy of the model developed. With reference to the results of classification, the performance of the classifier is determined.

Table 5.1 Precision and Recall values

S.no	Application	Precision	Recall
1	Bit Torrent	1.0	0.978
2	Facebook	0.76	1.0
3	Quake3	0.834	0.941
4	Skype	0.994	0.991
5	eDonkey	0.907	0.861

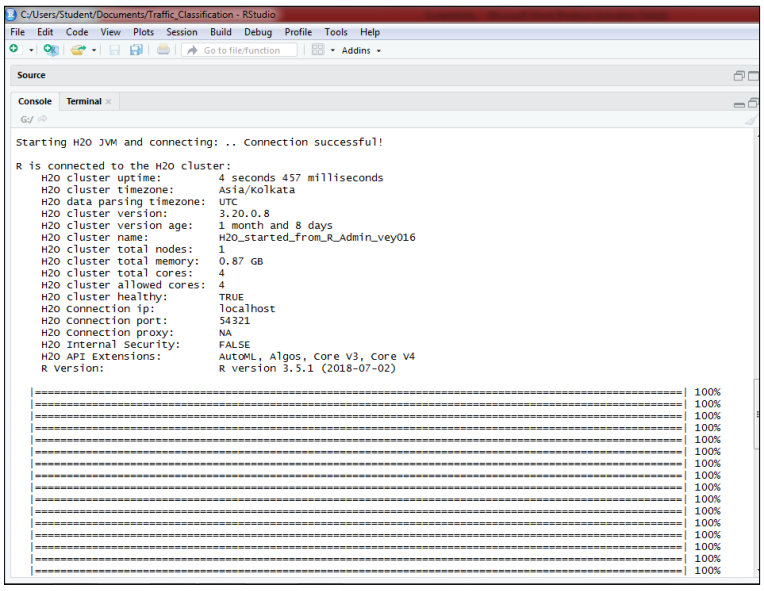


Fig 5.4 H2O cluster initialization for SAE

```
predict          BitTorrant          Facebook            GAME                Skype
Skype     :7808  Min.   :2.578e-12  Min.   :2.838e-10  Min.   :4.715e-14  Min.   :4.893e-05
eDonkey   : 511  1st Qu.:2.578e-12  1st Qu.:2.838e-10  1st Qu.:4.715e-14  1st Qu.:9.990e-01
BitTorrant: 440  Median :2.578e-12  Median :2.838e-10  Median :4.715e-14  Median :9.990e-01
GAME      : 404  Mean   :3.098e-02  Mean   :4.628e-03  Mean   :4.316e-02  Mean   :8.528e-01
Facebook  :   4  3rd Qu.:2.578e-12  3rd Qu.:2.838e-10  3rd Qu.:4.715e-14  3rd Qu.:9.990e-01
                 Max.   :7.031e-01  Max.   :4.054e-01  Max.   :9.898e-01  Max.   :1.000e+00
eDonkey
Min.   :4.518e-10
1st Qu.:4.518e-10
Median :4.518e-10
Mean   :6.838e-02
3rd Qu.:4.518e-10
Max.   :9.980e-01
```

Fig 5.5 Model built by SAE for evaluation

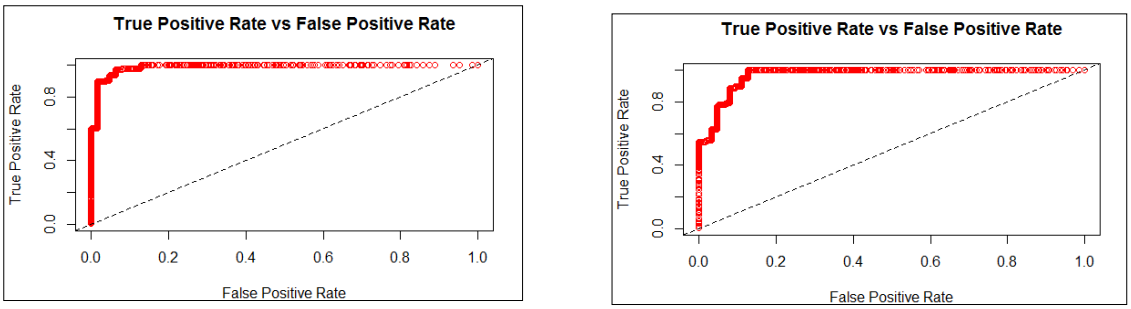


Fig 5.6 Performance of Skype and eDonkey application

Misclassification Table	(a)	(b)	(c)	(d)	(e)
(a)->BitTorrent					
(b)->Facebook					
(c)->Quake3					
(d)->Skype					
(e)->eDonkey					

Fig 5.7 Confusion Matrix

5.3 PERFORMANCE EVALUATION

Confusion matrix also called error matrix allows visualizing the performance of an algorithm. Performance of the system is also evaluated through precision and recall which is depicted in Table 5.1. The true positive and false positive values of Skype and eDonkey applications are compared in Fig 5.6. The confusion matrix for the classifier is depicted in the Figure 5.7. The overall accuracy was evaluated to be 98.195%. The value for the kappa coefficient, which takes into account chance occurrences of accurately classified flows and is generally considered a more robust measure than simple percent agreement calculation, was also significantly high at 0.919.

CHAPTER VI

CONCLUSION & FUTURE ENHANCEMENT

Network traffic classification, a hot area of research is facing trials due to the emergence of evading encryption and tunnelling techniques. The severity of attacks in P2P and VOIP application demanded potent deep analysis of traffic which is resistant to encryption techniques. This proposed system employed Stacked Auto Encoder, a deep learning based classifier with less complexity to build a model with discriminative features that is capable of evaluating the future network traffic with statistical features without decrypting them. The experimental results show that the system yields better results on the benchmark dataset CAIDA and UNB ISCX with accuracy of about 98.195%

The future work can be directed towards retraining point. Moreover accuracy of Facebook and Quake3 in the proposed system is only satisfactory and can be further enhanced by deeper analysis with multiple layers.

$$k = \frac{P_o - P_e}{1 - P_e}$$

APPENDIX

```
#include <stdio.h>
#include <getopt.h>
#include <errno.h>
#include <stdlib.h>
#include <pcap.h>
#include <netinet/in.h>
#include <net/ethernet.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/tcp.h>
#include <malloc.h>
#include <glib.h>
#include <string.h>

void pkt_callback(unsigned char *, const struct pcap_pkthdr *,
                  const unsigned char *);
int dump_remove_all(gpointer, gpointer, gpointer);

void free_key(gpointer);
void free_flowinfo(gpointer);

/* global variables */
pcap_t *p;
unsigned long int pkt_counter;
struct timeval ts_flushed;
char *flowdb = NULL;
int flowtimeout = 20;
GHashTable *active;
unsigned long int f_index = 1;
int f_dumped = 0;
unsigned long int fragdrops = 0;
unsigned long int fragdropsize = 0;
int lsec = 0;

struct f_info_t {
   unsigned long int index;
   unsigned long int bytes_in;
   unsigned long int bytes_out;
   unsigned long int pkts_in;
   unsigned long int pkts_out;
   struct timeval ts_first;
```

```
    struct timeval ts_last;
};

int main(int argc, char *argv[])
{
    extern char *optarg;
    extern int errno, optind;
    char *dumpfile = NULL;
    char *t_char;
    int t_int;

    /* process arguments */
    while ((t_int = getopt(argc, argv, "f:t:o:")) != EOF)
            switch (t_int) {
            case 'f':
                dumpfile = optarg;
                break;
            case 't':
                flowtimeout = atoi(optarg);
                if ((flowtimeout > 900) || (flowtimeout < 1)) {
                        fprintf(stderr,
                                "Flowtimeout should be between 1 and 900
seconds\n");
                        exit(1);
                }
                break;
            case 'o':
                flowdb = optarg;
                break;
            default:
                fprintf(stderr,
                        "Usage: %s -f <dumpfile> -o <outfile> [-t <flowtimeout>\n",
                        argv[0]);
                exit(1);
            }
    argc -= optind;
    argv += optind;

    if ((argc != 0) || (dumpfile == NULL) || (flowdb == NULL)) {
            fprintf(stderr,
                    "Usage: %s -f <dumpfile> -o <outfile> [-t <flowtimeout>\n",
                    argv[0]);
            exit(1);
```

```
    }

    fprintf(stderr,
            "Reading from %s with flowtimeout %d seconds\n", dumpfile,
            flowtimeout);

    /* open dumpfile */
    p = pcap_open_offline(dumpfile, t_char);
    if (!p) {
        fprintf(stderr, "Could not open dumpfile %s\n", t_char);
        exit(1);
    }

    /* initialize counters */
    pkt_counter = 0;
    ts_flushed.tv_sec = 0;
    ts_flushed.tv_usec = 0;

    /* create hash containing active flows */
    active =
        g_hash_table_new_full(g_str_hash, g_str_equal, free_key,
                              free_flowinfo);

    /* callback for all packets */
    if (pcap_loop(p, -1, &pkt_callback, NULL) < 0) {
        fprintf(stderr, "Error while processing dumpfile %s: %s\n",
                dumpfile, pcap_geterr(p));
        exit(1);
    }

    fprintf(stderr,
            "\n\nFinished processing dumpfile %s, processed %ld packets\n",
            dumpfile, pkt_counter);

    fprintf(stderr, "Dropped %lu fragmented packets, total %lu bytes\n",
            fragdrops, fragdropsize);

    g_hash_table_foreach_steal(active, dump_remove_all, NULL);


    exit(0);
}
```

```
void deltatime(struct timeval *result, struct timeval *high,
                struct timeval *low)
{
   /* set result intially to zero */
   memset(result, 0, sizeof(struct timeval));

   if (high->tv_sec == low->tv_sec) {
         result->tv_sec = 0;
         result->tv_usec = high->tv_usec - low->tv_usec;
   } else {
         result->tv_sec = high->tv_sec - low->tv_sec;
         if (high->tv_usec >= low->tv_usec) {
            result->tv_usec = high->tv_usec - low->tv_usec;
         } else {
            result->tv_sec -= 1;
            result->tv_usec = high->tv_usec - low->tv_usec + 1000000;
         }
   }

   return;
}

int check_flow_age(gpointer k, gpointer x, gpointer d)
{
   char *key = (char *) k;
   struct f_info_t *v = (struct f_info_t *) x;
   struct timeval *ts_cur = (struct timeval *) d;
   struct timeval res;

   /* check for aging flow */
   deltatime(&res, ts_cur, &v->ts_last);
   if (res.tv_sec >= flowtimeout) {
         void *k_orig;
         char *t_src_ip, *t_dst_ip, *t_proto, *t_src_port, *t_dst_port;


         k_orig = k;
         printf("\nFlow (index %lu)\n", v->index);
         printf("  key: %s\n", (char *) k);

         /* 'decode' key */
         t_src_ip = g_strndup((char *) k, 10);
         k += 10;
```

```
            t_dst_ip = g_strndup((char *) k, 10);
            k += 10;
            t_proto = g_strndup((char *) k, 2);
            k += 2;
            t_src_port = g_strndup((char *) k, 5);
            k += 5;
            t_dst_port = g_strndup((char *) k, 5);

            printf("  src ip: %llx\n", atoll(t_src_ip));
            printf("  dst ip: %llx\n", atoll(t_dst_ip));
            printf("  proto: %d\n", atoi(t_proto));
            printf("  src_port: %ld\n", atol(t_src_port));
            printf("  dst_port: %ld\n", atol(t_dst_port));

            printf("  bytes_in: %lu\n", v->bytes_in);
            printf("  bytes_out: %lu\n", v->bytes_out);
            printf("  bytes_total: %lu\n", v->bytes_in + v->bytes_out);
            printf("  pkts_in: %lu\n", v->pkts_in);
            printf("  pkts_out: %lu\n", v->pkts_out);
            printf("  ts_first: %ld %ld\n", v->ts_first.tv_sec,
                  v->ts_first.tv_usec);
            printf("  ts_last: %ld %ld\n", v->ts_last.tv_sec,
                  v->ts_last.tv_usec);
            deltatime(&res, &v->ts_last, &v->ts_first);
            printf("  duration: %ld %ld\n", res.tv_sec, res.tv_usec);

            free(t_src_ip);
            free(t_dst_ip);
            free(t_proto);
            free(t_src_port);
            free(t_dst_port);
            k = k_orig;


//      free(v);
//      free(k);
            f_dumped++;
            return (1);                 /* remove this one */
    }

    return (0);                         /* don't remove */
}
```

```
int dump_remove_all(gpointer k, gpointer x, gpointer d)
{
    struct f_info_t *v = (struct f_info_t *) x;
    struct timeval res;
    void *k_orig;
    char *t_src_ip, *t_dst_ip, *t_proto, *t_src_port, *t_dst_port;

    k_orig = k;
    printf("\nFlow (index %lu)\n", v->index);
    printf("  key: %s\n", (char *) k);

    /* 'decode' key */
    t_src_ip = g_strndup((char *) k, 10);
    k += 10;
    t_dst_ip = g_strndup((char *) k, 10);
    k += 10;
    t_proto = g_strndup((char *) k, 2);
    k += 2;
    t_src_port = g_strndup((char *) k, 5);
    k += 5;
    t_dst_port = g_strndup((char *) k, 5);

    printf("  src ip: %llx\n", atoll(t_src_ip));
    printf("  dst ip: %llx\n", atoll(t_dst_ip));
    printf("  proto: %d\n", atoi(t_proto));
    printf("  src_port: %ld\n", atol(t_src_port));
    printf("  dst_port: %ld\n", atol(t_dst_port));

    printf("  bytes_in: %lu\n", v->bytes_in);
    printf("  bytes_out: %lu\n", v->bytes_out);
    printf("  bytes_total: %lu\n", v->bytes_in + v->bytes_out);
    printf("  pkts_in: %lu\n", v->pkts_in);
    printf("  pkts_out: %lu\n", v->pkts_out);
    printf("  ts_first: %ld %ld\n", v->ts_first.tv_sec,
            v->ts_first.tv_usec);
    printf("  ts_last: %ld %ld\n", v->ts_last.tv_sec, v->ts_last.tv_usec);
    deltatime(&res, &v->ts_last, &v->ts_first);
    printf("  duration: %ld %ld\n", res.tv_sec, res.tv_usec);

    free(t_src_ip);
    free(t_dst_ip);
    free(t_proto);
    free(t_src_port);
```

```
    free(t_dst_port);
    k = k_orig;

//    free(v);
//    free(k);
    return (1);                         /* remove this one */
}

void free_key(gpointer d)
{
    char *k = (char *) d;
    free(k);
    return;
}

void free_flowinfo(gpointer d)
{
    struct f_info_t *v = (struct f_info_t *) d;
//    free(v);
    return;
}
void pkt_callback(unsigned char *t_char, const struct pcap_pkthdr *hdr,
                  const unsigned char *pkt)
{
    struct ether_header *ep;
    struct iphdr *ip;
    unsigned long int sport, dport;
    struct udphdr *udp;
    struct tcphdr *tcp;
    struct timeval t_ts;
    char *f_key;
    struct f_info_t *f_info, *t_val;
    int f_rev = 0;

    /* increment packet counter */
    pkt_counter++;

    /* discard non-IP packets */
    ep = (struct ether_header *) pkt;
    if (ntohs(ep->ether_type) != 0x0800)
        return;

    /* skip ethernet header */
```

```
    pkt += 14;
    ip = (struct iphdr *) pkt;

    /* discard non-UDP/non-TCP packets */
    if ((ip->protocol != 6) && (ip->protocol != 17))
            return;

    /* discard second, third, etc. fragments */
    if ((ip->frag_off & 0xff00) > 0) {
            fragdrops++;
            fragdropsize += hdr->len;
            return;
    }

    /* if this is the first packet, set time stamps */
    if (ts_flushed.tv_sec == 0) {
            ts_flushed.tv_sec = hdr->ts.tv_sec;
            ts_flushed.tv_usec = hdr->ts.tv_usec;
    }

    /* for every 1000th packet we print a dot and determine if flow dumping is
required */
    if ((pkt_counter % 1000) == 0) {
            fprintf(stderr, ".");

            deltatime(&t_ts, (struct timeval *) &hdr->ts, &ts_flushed);
            if (t_ts.tv_sec >= 1) {
               /* remove old flows */
               lsec += 1;
               fprintf(stderr,
                        "\n[%d] (time %lu %lu): ",
                        lsec, hdr->ts.tv_sec, hdr->ts.tv_usec);
               g_hash_table_foreach_steal(active, check_flow_age,
                                          (void *) &hdr->ts);
               fprintf(stderr, "%d dumped, %d active\n", f_dumped,
                        g_hash_table_size(active));
               ts_flushed.tv_sec = hdr->ts.tv_sec;
               ts_flushed.tv_usec = hdr->ts.tv_usec;
               f_dumped = 0;

            }
    }
    /* skip ip hdr */
```

```
pkt += ip->ihl * 4;

/* determine transport layer ports */
switch (ip->protocol) {
case 6:                         /* tcp */
        tcp = (struct tcphdr *) pkt;
        sport = ntohs(tcp->source);
        dport = ntohs(tcp->dest);
        break;
case 17:                        /* udp */
        udp = (struct udphdr *) pkt;
        sport = ntohs(udp->source);
        dport = ntohs(udp->dest);
        break;
}

/* construct char with flow key */
f_key = (char *) malloc(sizeof(char) * 33);
memset(f_key, 0, sizeof(f_key));
sprintf(f_key, "%.10lu%.10lu%.2d%.5ld%.5ld%s",
           (long unsigned int) ntohl(ip->saddr),
           (long unsigned int) ntohl(ip->daddr), ip->protocol, sport,
           dport, "\0");

/* lookup flowkey and reverse flowkey in hash table */
t_val = g_hash_table_lookup(active, f_key);
if (!t_val) {
        char *fr_key;
        fr_key = (char *) malloc(sizeof(char) * 33);
        memset(fr_key, 0, sizeof(fr_key));
        sprintf(fr_key, "%.10lu%.10lu%.2d%.5ld%.5ld%s",
                   (long unsigned int) ntohl(ip->daddr),
                   (long unsigned int) ntohl(ip->saddr), ip->protocol, dport,
                   sport, "\0");
        t_val = g_hash_table_lookup(active, fr_key);
        if (t_val) {
           f_rev = 1;               /* reverse flow */
           free(f_key);
           f_key = fr_key;
        } else {
           free(fr_key);
        }
}
```

```
    if (!t_val) {
        /* flow does not exist in active flow hash table */
        f_info = (struct f_info_t *) malloc(sizeof(struct f_info_t));
        memset(f_info, 0, sizeof(struct f_info_t));

        f_info->index = f_index++;
        f_info->bytes_out = hdr->len;
        f_info->pkts_out = 1;
        f_info->ts_first.tv_sec = f_info->ts_last.tv_sec = hdr->ts.tv_sec;
        f_info->ts_first.tv_usec = f_info->ts_last.tv_usec =
           hdr->ts.tv_usec;

        g_hash_table_insert(active, f_key, f_info);

    } else {
        /* flow exists in active flow hash table, update values */
        if (f_rev) {
           t_val->bytes_in += hdr->len;
           t_val->pkts_in += 1;
        } else {
           t_val->bytes_out += hdr->len;
           t_val->pkts_out += 1;
        }
        t_val->ts_last.tv_sec = hdr->ts.tv_sec;
        t_val->ts_last.tv_usec = hdr->ts.tv_usec;

        g_hash_table_replace(active, f_key, t_val);

    }

    return;
}
```

REFERENCES:

[1]Michael Cooney. “IBM warns of raising VOIP cyber-attacks”. Internet: https://www.networkworld.com/article/3146095/security/ibm-warns-of-rising-voip-cyber-attacks.html Nov30,2016[Nov16,2018].

[2] Lynette Lau. “Cybercrime 'pandemic' may have cost the world $600 billion last year”. Internet: https://www.cnbc.com/2018/02/22/cybercrime-pandemic-may-have-cost-the-world-600-billion-last-year.html. Feb.22, 2018[Nov.1, 2018].

[3] Aimee O’Driscoll.” 100+ Terrifying Cybercrime and Cybersecurity Statistics & Trends [2018 EDITION]. Internet: https://www.comparitech.com/vpn/cybersecurity-cyber-crime-statistics-facts-trends/. Oct.2, 2018[Nov.1, 2018].

[4] Chethan Kumar. “One cybercrime in India every 10 minutes”. Internet: https://timesofindia. indiatimes.com/india/one-cybercrime-in-india-every-10-minutes/articleshow/59707605.cms. Jul 22, 2017[Nov1, 2018].

[5] “Encrypted Traffic Analytics”. Internet: https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html. Dec, 2017[Nov7,2018].

[6] “Traffic Classification”. Internet: https://www.cisco.com/c/en/us/td/docs/nsite/enterprise/wan/wan_optimization/wan_opt_sg/chap05.html. Dec10, 2013 [Jan5, 2017].

[7] Thomas, Andre et al. “Transport Layer Identification of P2P traffic,” in *Proc. IMC ’04 ACM SIGCOMM Internet Measurement Conference*, 2004, pp. 121-134.

[8] Cross Validated.“Is there any difference between training a stacked autoencoder and a 2-layers neural network?,” https://stats.stackexchange.com/questions/134254/is-there-any-difference-between-training-a-stacked-autoencoder-and-a-2-layers-ne, Jan23, 2015 [Nov14,2018].

[9] Ralph Tkatchuk. “ Fighting malware with P2P cyber protection”. Internet: https://www.cio.com/article/3228744/cyber-attacks-espionage/fighting-malware-with-p2p-cyber-protection.html . Sep27,2017[Nov16,2018].

[10] Laurent Bernaille, Renata Teixeira et al. “Early Application Identification,” *in CONEXT’06, ACM,* 2006.

[11] Jeffrey Erman, Anirban Mahanti et al. “Offline/Real-time Traffic classification using Semi-Supervised Learning”, in *Performance Evaluation, Elsevier*, 2007.

[12] Thuy T.T.Nguyen, Grenville Armitage et al. “A survey of techniques for internet traffic classification in machine learning,” in *IEEE Communications Surveys & Tutorials*, 2008.

[13] Alberto Dainotti, Antonio Pescape et al. “Early Classification of Network Traffic through Multi-Classification,” in *Traffic monitoring and analysis, Springer,* 2011.

[14] Chun-Nan Lu, Chun-Ying Huang et al.” Session level flow classification by packet size distribution and grouping in internet traffic classification” in *Computer Networks, Elsevier,* 2011.

[15] Jun Zhang, Chao Chen et al. “An Effective Network Traffic Classification Method with Unknown flow detection,” in *IEEE Transactions on network & service management*, IEEE 2013.

[16] Jun Zhang, Chao Chen et al. “Internet Traffic classification by Aggregating Correlated Naïve Bayes Prediction,” in *IEEE transactions of information forensics and security* ,2013.

[17] Alina Vladutu, Dragos Comaneci et al. “ Internet traffic classification based on flows’ statistical properties” in *International Journal of Network Management,* Wiley Library 2016.

[18] Archanaa, Athulya et al. “A Comparative performance analysis on network traffic classification using supervised learning algorithms,” in *ICACCS,* IEEE 2017.

[19] Baris Yamansavascilar et al. “Application Identification via Network Traffic Classification,” in *CNC,* IEEE 2017.

[20] Giuseppe, Domenico et al. “Multi-Classification approaches for classifying mobile app traffic,” in *Preprint to Journal of Network and Computer Applications,* Elsevier 2017.