REAL-TIME ENCRYPTED TRAFFIC CLASSIFICATION WITH DEEP LEARNING

by

Deniz Tuana Ergönül

Submitted to Graduate School of Natural and Applied Sciences
in Partial Fulfillment of the Requirements
for the Degree of Master of Science in
Computer Engineering

Yeditepe University

2021

REAL-TIME ENCRYPTED TRAFFIC CLASSIFICATION WITH DEEP LEARNING

APPROVED BY:

Assist. Prof. Dr. Onur Demir .....
(Thesis Supervisor)
(Yeditepe University)

Prof. Dr. Fatih Uğurdağ .....
(Özyeğin University)

Assist. Prof. Dr. Tacha Serif .....
(Yeditepe University)

DATE OF APPROVAL: .... /.... /2021

I hereby declare that this thesis is my own work and that all information in this thesis has been obtained and presented in accordance with academic rules and ethical conduct. I have fully cited and referenced all material and results as required by these rules and conduct, and this thesis study does not contain any plagiarism. If any material used in the thesis requires copyright, the necessary permissions have been obtained. No material from this thesis has been used for the award of another degree.

I accept all kinds of legal liability that may arise in case contrary to these situations.

Name, Last name Deniz Tuana Ergönül

Signature .....

ACKNOWLEDGEMENTS

Assist. Prof. Dr. Onur Demir, my thesis supervisor, was very much supportive throughout the whole process despite my indecisiveness and change of hearts; thus, I owe him a big thank you for his understanding.

I want to mention my dear colleagues from NETTSI Informatics Technology, especially Aslıhan Reyhanoğlu, for sharing countless research papers and holding mini brainstorming sessions with me, and Serkan Eser, for being an inspiring mentor whom I will always look up to.

Special thanks to my partner, Arif Orhun Uzun, who spent his time dwelling upon some of the problems I encountered, giving me tips and tricks to get back on track.

Last but not least, I would like to thank my family for their support.

ABSTRACT

REAL-TIME ENCRYPTED TRAFFIC CLASSIFICATION WITH DEEP LEARNING

With the widespread use of encryption, and VPN (Virtual Private Network) usage increase, traffic classification became difficult. It provides a way to take certain actions for specific traffic (e.g., different pricing, creating a safe internet for children) and utilize resources to be optimally used according to traffic. It engages the attention of internet providers, and governments. Apart from traditional methods: port-based, signature-based, and statistical; machine learning (ML), and deep learning also started to become popular. When encrypted, traffic can be harder to classify as packet content becomes unreadable. This study gains an advantage for encrypted traffic as it does not examine payload. Most of the work done used pre-collected packets, limits of real-time classification are not visible. This work aims to contribute to the shaping of these boundaries. Accuracy and packet processing time are on the radar. LSTM (Long Short-Term Memory) is a good candidate for this problem as it can handle sequences. Each flow can be modeled as a sequence. By adapting one of the studies in field, an ML model trained with statistical features is presented along with a new LSTM model. Compared to other LSTM studies, packets are not discarded if their flow is longer than the preset sequence length. Features are extracted from packet headers only. 14 labels are used to test the proposed solutions in total: non-VPN, VPN, 6 non-VPN categories, 6 VPN categories. Tests showed that LSTM is valid for traffic categorization in terms of accuracy and speed. Compared to the reference ML method, LSTM excelled with precision and recall differences up to 50 percent. The adapted algorithm is more accurate than the original. Accuracy with LSTM was measured as 97.77 percent offline and 91.7 in real-time. Packet processing time was recorded as 0.593 ms which is 5 times faster than another LSTM method. Flow-based ML has an accuracy of 99.83 percent, while packet-based has 99.99.

ÖZET

DERİN ÖĞRENME İLE GERÇEK ZAMANLI ŞİFRELİ TRAFİK SINIFLAMA

Şifreleme algoritmalarının yaygınlaşması ve VPN (Sanal Özel Ağ) kullanımının artmasıyla trafik sınıflama zorlaştı. Sınıflama sayesinde spesifik trafikler için belirli aksiyonlar alınabiliyor (Bkz. farklı fiyatlandırmalar, çocuklar için güvenli internet), ağdaki trafiğe göre kaynaklar optimal şekilde kullanılmak üzere ayarlanabiliyor. Bu yönleriyle özellikle internet sağlayıcılarının ve devletlerin ilgisini çekmektedir. Port tabanlı, imza tabanlı, istatiksel metotlara dayalı geleneksel sınıflama yöntemlerinin dışında makine öğrenmesi (ML) ve derin öğrenme de popülerleşti. Trafik şifreli olduğunda paket içeriği okunamaz hale geldiği için sınıflama yapmak zorlaşabiliyor. Bu çalışma, paket yüküne bakmaması yönüyle şifreli trafik sınıflamada avantajlı. Yapılan çalışmaların çoğu, önceden toplanan paketlerle yapılmış olup gerçek zamanlı sınıflamanın sınırları görülememektedir. Bu çalışma, paket akışlarının kategori bazında sınıflanmasında bu sınırların şekillenmesine katkıda bulunmayı hedefler. Doğruluk ve paket işleme süresi bu çalışmanın kriterlerindendir. LSTM (Uzun Kısa Süreli Bellek) sekanslarla çalışabilmesiyle bu problem için uygun bir adaydır. Her paket akışı bir sekans olarak modellenebilir. Çalışmada, bu alandaki araştırmalardan biri uyarlanarak geliştirilen, istatiksel özniteliklerle eğitilen bir ML ve yeni bir LSTM modeli sunuldu. Geçmiş LSTM çalışmalarından farklı olarak, belirlenen sekans uzunluğunu aşan akışların paketleri ekarte edilmez. Öznitelikler yalnızca paket başlıklarından çıkarılır. Test için toplam 14 kategori kullanıldı: VPN olmayan, VPN, 6 VPN olmayan kategori, 6 VPN kategorisi. Testler, trafik kategorizasyonu için LSTM yaklaşımının doğruluk ve hız açısından geçerli olduğunu gösterdi. Referans makine öğrenimi yöntemiyle kıyaslandığında, LSTM, %50'ye varan doğruluk farklılıklarıyla öne çıktı. Uyarlanan algoritma doğruluk açısından orijinaline göre daha iyi sonuçlar verdi. LSTM ile doğruluk çevrimdışında 97.77%, gerçek zamanda 91.7% olarak ölçülmüştür. Paket işleme süresi 0.593 ms ile farklı bir LSTM yönteminden 5 kat daha hızlı olarak kayda geçmiştir. Akış-bazlı ML 99.83%, paket-bazlı ise 99.99% doğruluğa sahiptir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

λ	Rate parameter of exponential distribution
μ	Mean
σ	Standard deviation
$\overline{x}$	Mean of x
1-D	One-dimensional
AIM	AOL Instant Messenger
AOL	America Online
ARFF	Attribute-relation file format
ASCII	American Standard Code for Information Interchange
CNN	Convolutional neural network
DNS	Domain Name System
DPI	Deep packet inspection
FTP	File Transfer Protocol
FTPS	FTP Secure
GRU	Gated recurrent unit
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IP	Internet Protocol
ISCX	Information Centre of Excellence for Tech Innovation
ISCXVPN2016	UNB ISCX network traffic dataset
ISP	Internet service provider
k-NN	k-nearest neighbors
LSTM	Long short-term memory
LSTM-FS	LSTM with flow sequences
ML	Machine learning

ML-SF	Machine learning with statistical features
MLP	Multi-layer perceptron
NLP	Natural language processing
NN	Neural network
P2P	Peer-to-peer
PCAP	Packet capture
QUIC	Quick UDP Internet Connections
RNN	Recurrent neural network
SMTP	Simple Mail Transfer Protocol
SPID	Statistical Protocol Identification
SSH	Secure Shell
SSL	Secure Sockets Layer
SVM	Support vector machine
TCP	Transmission Control Protocol
TLS	Transport Layer Security
Tor	The Onion Router
UDP	User Datagram Protocol
UML	Unified Modeling Language
UNB	University of New Brunswick
VPN	Virtual private network
VoIP	Voice over IP

1. INTRODUCTION

This chapter contains two sections: Motivation (Section 1.1) and Aim (Section 1.2). The Motivation section explains the problem investigated in this thesis and presents essential information about the relevant domain. In addition, this section mentions why it is worth examining, the factors that complicate it, and how it has been solved so far. Contributions of this study to the domain are shared in the Aim section. Its key differences from the solutions tried before are emphasized.

1.1. MOTIVATION

Traffic classification is the task of labeling network packets according to some criteria desired by the user [2]. Identifying the class of a network packet can be used for running predefined rules on it, such as ensuring the quality of service promised to the customer [3], detecting malicious user activities, or dropping certain content for parental control [4]. Classification can be in terms of the application, protocol, or category of a packet; for example, WhatsApp Web application, HTTPS protocol, chat category; Skype application, TLS protocol, VoIP category [5]. Specific use case examples of traffic classification in real life are: offering specialized internet packages according to customer needs (prioritizing online gaming traffic and reducing latency in the game for an online gamer), setting a special discount for social media platforms (e.g., Facebook, Twitter), or managing traffic of protocols that use too much bandwidth such as BitTorrent. Mostly, it is the focus of ISPs (Internet Service Provider) and defense industry units of countries [6]; however, every institution can benefit from a dedicated solution according to its purpose. Labeling of pre-captured network packets is called offline classification. Online classification is when they are labeled one after another in real-time [7, 8].

Classification is usually performed on bidirectional flows instead of a single packet. A flow is a set of packets exchanged between two distinct networks, using the same transport protocol (e.g., TCP, UDP) and port numbers. What identifies a flow is called a five-tuple: transport protocol, source IP, destination IP, source port, destination port. In a bidirectional

flow, packets that have their IP and port pairs reversed also counts. Once a flow is labeled, any packet that belongs to that flow can be labeled as the same.

The oldest and easiest method of classifying a flow is querying for its port numbers in IANA's (Internet Assigned Numbers Authority) Service Name and Transport Protocol Port Number Registry [9]. Although practical in respect of time and effort, this technique is not very reliable. Because, there is no restriction for protocols to use only the standard ports. There is even a term called port obfuscation [10]. Protocols using obfuscation (e.g., VPNs) willingly avoid using the standards to hide their identity.

A somewhat more advanced way to classify flows is to do deep packet inspection (DPI), in other words, to analyze all layers of packets, including packet payload. An unencrypted payload can give a clear idea of what protocol a packet uses. Some protocols have unique patterns (signatures) which can be directly used to label packets. Regular expression databases are generated to store this information. However, maintaining such a database is a tedious job. On the other hand, payload encryption got popular over the years. As an example, Google Transparency Report [11] (Fig. 1.1) shows that the preference for HTTPS over HTTP has increased significantly in the last seven years [12], from 48 percent in 2014 to 95 percent in 2021. Encryption changes the content of packets in such a way that they are no longer human-readable. For this reason, it becomes very challenging to find an easily recognizable pattern in the payload.

Statistical flow classification is another valid method that uses features such as average packet inter-arrival times, packets per second, byte frequencies, etc [13].

In recent years, machine learning and deep learning models became preferable, especially when there is enough data for training. The advantages of these models are that they can be trained offline but tested online, and they provide very high accuracy rates. As it is for many learning problems, feature selection may be considered a disadvantage, but it is possible to make a model learn how to do that on its own. Deep learning is a relatively new term for flow classification. There are not many research papers yet, especially with an LSTM model, and the existing ones usually do offline classification [14]. Thus, the elapsed time,

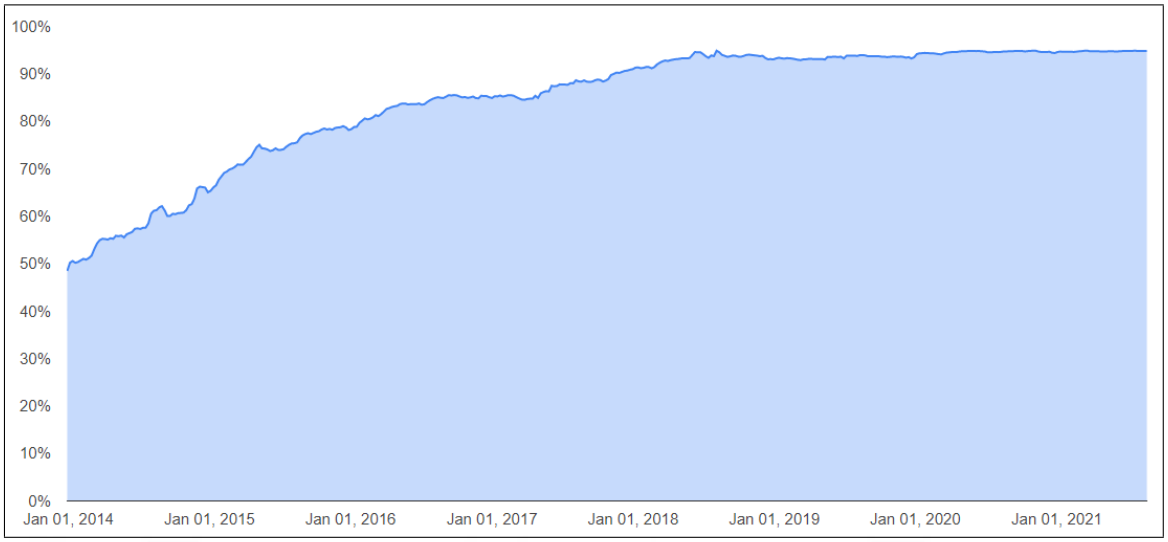

Figure 1.1. Encrypted traffic across Google

a critical performance measure for online classification, does not get mentioned. One of the challenges of traffic classification is data, as it is for almost all learning problems. Tools such as Wireshark [15] can collect live traffic into packet capture (PCAP) files. The difficulty here is that the collected data is often noisy. The operating system, services running in the background, other applications, or services that applications use might be the reason for the noise. It is crucial to feed learning models with clean data since any noise might cause them to make incorrect connections, leading them to deduct unreliable knowledge. Thankfully, there is a very common public dataset named ISCXVPN2016 (UNB ISCX Network Traffic Dataset) [16] that is used prevalently in this area. However, it is worth noting that a model trained with data obtained from a particular network may not perform well on a different network [17].

1.2. AIM

This work aims to classify flows, even if they contain encrypted packets, as to their categories, in real-time. Encryption adds a new difficulty level since the packet payload becomes indecipherable to infer a piece of information from it without decryption. This difficulty has no effect on the proposed solution because the solution does not require the examination of packet payloads. Crucial metrics for this work include accuracy, which is

the most emphasized performance criterion, and elapsed time for processing (analyzing and classifying) a single packet. The biggest goal of the study is to examine the performance of an LSTM model named LSTM-FS (LSTM with Flow Sequences) for the categorization problem both offline and online, to find out accuracies, to observe the number of packets processed per second in the online environment, and the processing time of a single network packet. The secondary aim of the study is to adapt a machine learning method named ML-SF (Machine Learning with Statistical Features) to improve this method.

Here is the list of questions this study intends to shed some light on:

- Why is LSTM a good candidate for traffic classification problem?
- How to model this problem for LSTM?
- How does LSTM perform both accuracy and time-wise?
- Could current ML-SF results be improved?

LSTM, a variant of RNN (Recurrent Neural Network), can store information for a while and use this information for future predictions. It has structures called gates which let information be stored or forgotten. This approach allows LSTM to perform well with data sequences. For instance, LSTM is used for NLP (Natural Language Processing) problems because words in a paragraph are somehow related, and LSTM can figure out these relationships. Similarly, network packets also have temporal information thus can be handled in sequences. Rather than being independent, consecutive packets in a flow are semantically related to each other. Packets in a flow are not random, as words in a sentence are not random. The reason for choosing LSTM is that it has a memory that can be utilized to store useful information of earlier packets of a flow. What is inside an encrypted network packet is often a mystery; it is not straightforward how to obtain information from them with techniques such as DPI. It is both challenging and time-consuming for the human eye to find consistent patterns for each label when payloads are encrypted. However, LSTM can make deductions from data that are not directly visible. The machine learning method also has an advantage over DPI; because it examines packets only superficially to produce statistics of packet lengths and inter-arrival times between them.

The difference between the presented LSTM implementation and earlier ones is that if a flow has more packets than the predefined sequence length (10), instead of discarding packets, new sequences are created. Besides, none of the proposed 17 features include information from packet payload; all are extracted from packet headers.

In total, 14 classification labels are used among 4 scenarios to test LSTM-FS and ML-SF: non-VPN, VPN, chat, email, file transfer, P2P, streaming, VoIP, VPN chat, VPN email, VPN file transfer, VPN P2P, VPN streaming, and VPN VoIP. The first scenario detects whether the traffic is VPN or not, using the labels VPN and non-VPN. The second scenario contains PCAPs of 6 non-VPN categories: chat, email, file transfer, P2P, streaming, and VoIP. Similarly, the third scenario only includes PCAPs of 6 VPN categories. In the last scenario, with the second and third scenarios combined, labeling is made among 6 non-VPN and 6 VPN categories. Along with offline tests, online simulations also exist for each scenario of LSTM-FS. LSTM-FS gave good results with respect to speed and accuracy. For ML-SF, both flow-based and packet-based methods are tried. ML-SF got better accuracies than the original study on which it is based.

Chapter 2 includes briefs of 11 studies in the traffic classification field. At the end of it, there is a table that summarizes the big picture. The presented work is explained in Chapter 3 in full detail. The program involves implementations of both LSTM-FS and ML-SF. The chapter contains a UML (Unified Modeling Language) class diagram, plus tables describing the fields and functions of the classes. Chapter 4 introduces the environment in which the tests ran, the test scenarios, the data and its usage, and the parameters; it contains test results. Inferences made following the tests are shared in Chapter 5; possible improvements and new ideas for future work are also mentioned.

2. BACKGROUND

This chapter contains summaries of 11 previous studies. In the Summary section, a general evaluation is made about these studies; and those that are closely related to the proposed work are discussed separately for comparison. 1 of the 11 studies is a purely statistical method, while 3 of them consist of machine learning models trained with statistical features. There is 1 survey paper about deep learning methods that also underlines the difficulties of encrypted traffic classification. The remaining 6 are deep learning methods, 2 of which are LSTM. They classify protocols, applications, or categories. Only the last study mentions real-time classification. ISCXVPN2016 [16] stands out as the dataset used.

Statistical Protocol Identification (SPID) [18, 19, 20] is an open-source project that accepts PCAP files as input to generate protocol models by examining flow and payload statistics. Before any classification can be performed, the user must initiate training by feeding pre-labeled PCAP files to SPID. At the training phase, for each flow in a pre-labeled PCAP, a protocol model is generated. SPID keeps generated models in a database. If there exists another model with the same label, the two models get merged. At the testing phase, a new model is generated that corresponds to the current session; then, it is compared with the models in the database. SPID algorithm uses packets with payload in a session to calculate some features (attribute meters). A protocol model is made of these meters. There are 34 available, 7 of them are flow-level, and 27 are payload-level. Each has a counter vector of size 256 that has a unique meaning for the meter. For example, ByteFrequency is a payload-level meter that interprets this vector as a list of ASCII characters where index 65 corresponds to the 'A' character, for instance. Each meter calculation returns a list of indices for the meter's counter vector. The elements at these indices is increased by 1. The researchers got an average of 100 percent precision and 92 percent recall when tested with BitTorrent, eDonkey, HTTP, SSL, and SSH flows.

Characterization of Encrypted and VPN Traffic Using Time-Related Features [1] is a significant study for two reasons: they provide a public dataset named ISCXVPN2016 which has been used by many researchers through the years, and they present a machine learning

approach for flow classification. The dataset consists of various PCAPs, and it is 26.2 GB large. The files can be grouped into 14 different categories: chat, email, file transfer, P2P, streaming, VoIP, VPN chat, VPN email, VPN file transfer, VPN P2P, VPN streaming, VPN VoIP, Tor (The Onion Router) browsing, Tor streaming. They calculate a total of 24 time-based, statistical features to perform classification. The main features include flow duration, the time between two successive packets (in the forward, backward, and both directions), the time a flow is active and idle, flow bytes per second, and flow packets per second. Other features are the minimum, maximum, average, and standard deviation of these features. They employed four flow timeout values 15, 30, 60, and 120; to find out 15 is the most accurate. They had accuracy levels above 80 percent.

Yamansavascilar et al. [17] used machine learning algorithms J48 (also known as C4.5), Random Forest, k-Nearest Neighbors (k-NN), and Bayes Net to identify popular consumer applications such as; Facebook, Twitter, and Skype. Just as in this paper, the ISCXVPN2016 dataset is made use of along with an internal dataset. For applications that are available on mobile phones, they captured cellular traffic as well. The datasets contain 15,462 and 3,748 flows, respectively. They chose 111 features, how they chose these features seems unclear. Accuracy varies between 85.44-93.94 percent for the external dataset and 69.90-90.87 percent for the internal. Using evaluators, they succeeded in decreasing the number of features as down as 12; while maintaining the accuracy levels. They realized that the misclassified samples are often in the same category. Therefore, category classification before application identification can be considered as future work.

Bagui et al. [21] compared the performance of six supervised machine learning techniques Logistic Regression, Support Vector Machine (SVM), Naive Bayes, k-NN, Gradient Boosting Trees, and Random Forest, in terms of accuracy, precision, sensitivity, and specificity. They utilized the same 24 time-related features in [1] along with the ISCXVPN2016 dataset. Separately for each category (browsing, chat, email, file transfer, P2P, streaming, VoIP), they did binary classification to identify if traffic is VPN encrypted or non-VPN encrypted. The sample distribution within each category is well-balanced (almost 50 percent-50 percent). Sample amounts range between 1,113 and 10,000, not enough to try out deep neural networks, as they state. Gradient Boosting Tree and Random Forest

outperformed the other methods in respect of all metrics with values around 94 percent. Aiming for high accuracy and low overfitting, they used grid search to find an optimal, work-for-all set of hyperparameters which increased the values by a few percent. Additionally, through feature selection, they ordered a minimum set of features by importance for each category. Although feature selection worsens accuracy results to 90 percent, they claim that it can be advantageous where speed matters more, but they did not present any experiment results regarding this assumption.

One-dimensional Convolutional Neural Network (1-D CNN) is an apt model for traffic identification and classification, Wang et al. [22] claimed, as network traffic is sequential data. They argue that solution becomes more prone to get stuck at a local minimum when a problem is divided into sub-problems. So, they present an end-to-end solution where feature extraction, feature selection, and classifier are one [23]. In other words, their model takes raw data and learns features on its own. They used the ISCXVPN2016 dataset, excluded capture files that are vague. (e.g., Both browsing and streaming is a suitable label for "Facebook_video.pcap", they say.) To do data pre-processing, they adopted a toolkit of their team named USTC-TK2016 [24] which transforms capture files into images. Wang et al. noticed that the images generated for same-class flows looked very similar. They observed that bidirectional flows yield higher accuracy than unidirectional ones; and as opposed to using only the application layer of a packet, using all layers is better. Since CNN accepts equally long data, they employed only the first 784 bytes of each flow. How many samples they had for each category is not given in the paper. However, they did mention an imbalance in the dataset as a future concern. They did four experiments: binary classification between VPN and non-VPN data, category classification among non-VPN, VPN, and mixed data. Best accuracy results were respectively 99.9, 83, 98.6 and 86.6 percent. They also pointed out that 1-D CNN delivers better performance than two-dimensional.

Rezaei and his team [14] provided a survey paper about traffic classification with deep learning techniques, namely, Multi-Layer Perceptron (MLP), CNN, RNN, Autoencoder, and Generative Adversarial Network. They emphasized the challenging aspects of this field, such as data collection. Other than ISCXVPN2016 and a few others, there is no publicly available and widely used dataset, they state. Researchers tend to capture data on their own,

specific to their needs. Because of background traffic caused by routers, operating systems, et cetera, collecting noiseless capture files is far from an easy task. Another issue is that a model trained with a particular network's data might perform way worse when tested on another network. Or even on the same network but under different conditions (i.e., high congestion). They also stressed how to tackle encrypted traffic. For example, time-series features such as inter-arrival time are not affected by encryption; because they do not depend on packet payloads. Using the payload of the first TCP packets may still be convenient since the handshake stays unencrypted. However, the authors also pointed out that newer encryption protocols QUIC and TLS 1.3 remain open for investigation. Other problems they discussed are zero-day applications (classes the model has not seen yet), multi-label classification (one flow carrying multiple classes), middle flow classification (classification using packets from the middle of the flow), transfer learning, and multi-task learning.

Deep Packet [25] can identify the protocol or application to which a packet belongs (by 98 percent recall), label it by its category while differentiating VPN from non-VPN (94 percent). Lotfollahi et al. used the ISCXVPN2016 dataset by dividing it into 17 application classes for the first experiment; 6 VPN, 6 non-VPN category classes for the second. Instead of flows, they work with packets, and the packet distribution within each set of classes is highly imbalanced. (e.g., 5K AIM vs. 7872K FTPS , 13K VPN email vs. 5120K VoIP) To handle this matter, they did under-sampling; they excluded samples from dominant classes. Their model consists of two parts: stacked autoencoders and 1-D CNN. The autoencoders take care of feature extraction automatically, eliminating the need for an expert. Since inputs fed into a NN must be fixed-length, they decided on 1500 bytes after examining packet length histogram. Some pre-processing steps are padding UDP headers (so they become equal length with TCP), discarding TCP handshake and DNS packets (bring no useful information), and masking IP addresses (causes over-fitting). They explain how their model can classify encrypted packets with the following hypothesis: although encrypted, packets of the same application may still contain consistent patterns. This hypothesis is also how they justify why their model failed to classify Tor categories.

Parchekani et al. [26] adopted the ISCXVPN2016 dataset and a local ISP's data to classify five non-VPN (chat, email, FTP, streaming, VoIP), and one VPN class which symbolizes all

kinds of VPN traffic. They fed each flow's first 784 bytes into their models. The models consist of MLP and RNN layers. To receive the best precision values (above 80 percent), they tried two approaches: score and distance. Their study reveals that distance is a better metric than score. Both include a definition of a threshold parameter to either accept or reject a label. For the given input, in the first approach, each candidate non-VPN label gets a score. If the one with the maximum score is less than the threshold, it gets rejected. In the second approach, candidates get a distance value; and if the label with the minimum distance is more than the threshold, it is not accepted. In both cases, rejection means classification as VPN. Accepted ones go through a second phase where their category decision becomes finalized. To achieve this, they use other thresholds to decide if another classification is necessary.

With his team, Zhou [27] combined entropy estimation with traditional machine learning methods SVM, Random Forest, Naive Bayes, and Logistic Regression along with a NN, to distinguish VPN from non-VPN on the ISCXVPN2016 dataset. Compared with metrics from past work, results improved by 1 percent to 7 percent, with Random Forest remaining the best method (98 percent). Additionally, they built a NN which takes 23 statistical features (from [1]) as input to classify Tor data (ISCXTor2016 [28]) into 8 categories, as opposed to 7 seen in prior work. (Streaming is divided into two labels: audio and video) With this model, they improved some of the previous studies' metrics by nearly 30 percent. They also examined the 23 features using principal component analysis and found that it is possible to eliminate half of the features without significantly sacrificing accuracy.

Network traffic classifier by Lopez-Martin et al. [29] integrated LSTM on top of CNN to classify a dataset from RedIRIS [30], which contains 266,160 flows with 108 applications. They used nDPI [31], an open-source DPI library, to label the data. Label distribution is quite imbalanced; almost half of the data is HTTP, followed by DNS with 20 percent and SSL with 15 percent. After trying out a few combinations, they settled on five features: source port, destination port, payload length, TCP window size, and packet direction. (96 percent accuracy) They observed that adding timestamp as a feature did not enhance the results. Before further experiments, they set the sequence length as 20 packets. For each flow, they discarded the packets after the 20th; and if the flow was not long enough, they

applied padding. Later they realized that a length of 5 to 15 is sufficient for good results. Among models CNN-only, RNN-only, and CNN with LSTM, the latter performed the best with an accuracy of 96 percent.

Byte Segment Neural Network is a model by Liu et al. [32] based on RNN variants LSTM and GRU (Gated Recurrent Unit) to classify protocols and applications DNS, BitTorrent, PPLive, QQ, SMTP, 360, Amazon, Yahoo, Cloudmusic, and Foxmail. The real-world data they used consisted of 2,516 (DNS) to 55,576 (Cloudmusic) samples/datagrams. All header information (Ethernet, IP, TCP/UDP) got removed from datagrams. The segment generator splits the payload into equally-lengthed byte segments. According to their experiments, the most practical length is 8, against 3, 5, and 10. Compared to nDPI and a binary classifier named Securitas [33], they got better results for 5 of the applications considering recall, precision, and F-score. They have an average F-score of 95.82 percent. In the training phase, it takes 43 ms on average to prepare input, and 20 epochs of training take 123 minutes, whereas, in the testing phase, processing a datagram takes 2.97 ms.

2.1. SUMMARY

Table 2.1 consists of all the studies summarized except Deep Learning for Encrypted Traffic Classification [14], a survey paper. Classification is usually done based on protocol, application, or category. The methods used include purely statistical methods as well as decision tree solutions with statistical features. Tree-based machine learning methods have gradually started to be replaced by deep learning methods. Features used include both flow-level and payload-level information. Specifically, the use of time-based features is quite common. All studies except the last take place offline.

Unquestionably, the work of Draper-Gil et al. stands out in this area. Noise-free data is a big problem in traffic classification as in almost every field; Draper-Gil and his team present a quality dataset that is used by many researchers after them. This dataset is quantitatively full enough to be used even for deep learning solutions; plus, it consists of popular applications (such as Youtube, Netflix, Skype, etc.). Besides, it can easily be seen that the 24 time-based features they offer inspired many studies; ML-SF is one of these work. It could be helpful

if they shared how they labeled the data on a category-by-category basis; so that the results could be better evaluated. Thus, we could fully understand why we achieved better results even though we used the same features in flow-based ML-SF. The only visible difference between the two studies is that they have an additional label named browsing.

The five features used by the penultimate study, one of the 2 studies that use LSTM, are taken directly from the packets as in this study (LSTM-FS). Unlike that study, there are 23 features defined in LSTM-FS. They use flows of length 20 as sequences, any flow longer gets its packets discarded, and any flow shorter gets padded. On the contrary, the length of 10 is used in LSTM-FS, and no packets get discarded; instead, they are handled in new sequences. The other LSTM-based study only checks payloads, which is different than in LSTM-FS. They are the only researchers who shared elapsed time results. They report 2.97 ms as the average datagram processing time; our proposed solution LSTM-FS performs 5 times faster than their solution.

Table 2.1. Properties of prior work

Cite	Paper	Classification	Method	Features
[20]	Statistical Protocol IDentification with SPID: Preliminary Results	Protocol	Statistical	7 flow-level, 27 payload-level
[1]	Characterization of Encrypted and VPN Traffic Using Time-Related Features	Category	k-NN, C4.5 (J48)	24 time-based
[17]	Application Identification via Network Traffic Classification	Application	J48, Random Forest, k-NN, Bayes Net	111 (Reduced to 12)
[21]	Comparison of machine-learning algorithms for classification of VPN network traffic flow using time-related features	Category	Logistic Regression, SVM, Naive Bayes, k-NN, Gradient Boosting Trees, Random Forest	24 time-based
[22]	End-to-end Encrypted Traffic Classification with One-dimensional Convolution Neural Networks	Category	1-D CNN	First 784 bytes of a flow
[25]	Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning	Protocol, Application, Category	Stacked autoencoders + 1-D CNN	First 1500 bytes of a packet
[26]	Classification of Traffic Using Neural Networks by Rejecting: a Novel Approach in Classifying VPN Traffic	Category	MLP + RNN	First 784 bytes of a flow
[27]	Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks	Category	Entropy estimation + (SVM, Random Forest, Naive Bayes, Logistic Regression, NN)	23 time-based
[29]	Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things	Protocol, Application	CNN, RNN, CNN + LSTM	5
[32]	Byte Segment Neural Network for Network Traffic Classification	Protocol, Application	LSTM, GRU	Equally-lengthed byte segments

3. METHODOLOGY

In this chapter, the architectural structures of the two solutions proposed for encrypted traffic classification, LSTM-FS (LSTM with Flow Sequences) and ML-SF (Machine Learning with Statistical Features), are explained. The class details of the two projects designed with object-oriented philosophy are shared. The fields and functions of the classes, association, composition, and aggregation relationships are shown along with their hierarchical order.

PcapAnalyzer and FlowStatisticsCalculator [34] are two projects under the solution that is being proposed in this thesis. FlowStatisticsCalculator project involves flow generation and statistics calculation to extract the features defined in [1]. It is a C++ adaptation of ISCXFlowMeter [35], although it may not be an exact match. Both aim to detect the packets with the same endpoints and ports so that they are associated with the same flow; then generate statistics for the flows. PcapAnalyzer includes the implementation of LSTM-FS, and FlowStatisticsCalculator includes ML-SF. The UML diagram in Fig. 3.1 covers our entire solution, presenting the classes with their fields and functions, and the relationships between the classes.

Both PcapAnalyzer and FlowStatisticsCalculator work with PCAP files; and include libpcap [36], a C/C++ library for network traffic capture operations. Wireshark [15], which is a well-known open-source network packet analyzer, also utilizes this library. Both projects support Ethernet as data link layer (Layer 2, L2) protocol; IPv4 as network layer protocol (Layer 3, L3); TCP and UDP as transport layer (Layer 4, L4) protocol. Raw IP packets which do not involve a data link layer at all are not supported directly. Instead, these PCAPs are modified with a tool named tcprewrite [37] so that their packets have fake Ethernet layers.

It may not be straightforward what is meant by packet length, which is a common keyword for both projects. Valid candidates are length on wire (actual length, frame length), captured length, IP total length, and TCP or UDP length. The first two got crossed out since Ethernet packets might get padded. IP length seems to be the best choice since it carries higher layer lengths as well. It is also important to note that packets lined up in a PCAP file may not

represent their actual order. That is why epoch time is a more reliable reference since it indicates when a packet is captured. Another complication might be some flows having identical five-tuple but belonging to different labels; as a temporary solution, only the first seen flow counts.

Diagrams, tables, and algorithms in this chapter might contain more representable, simplified namings than the original code. Too specific implementation-related details are omitted as much as possible to keep the explanation easy to read and understand. Even though not strictly followed, the UML specification of OMG (Object Management Group) [38] is used as a guide for the diagrams.

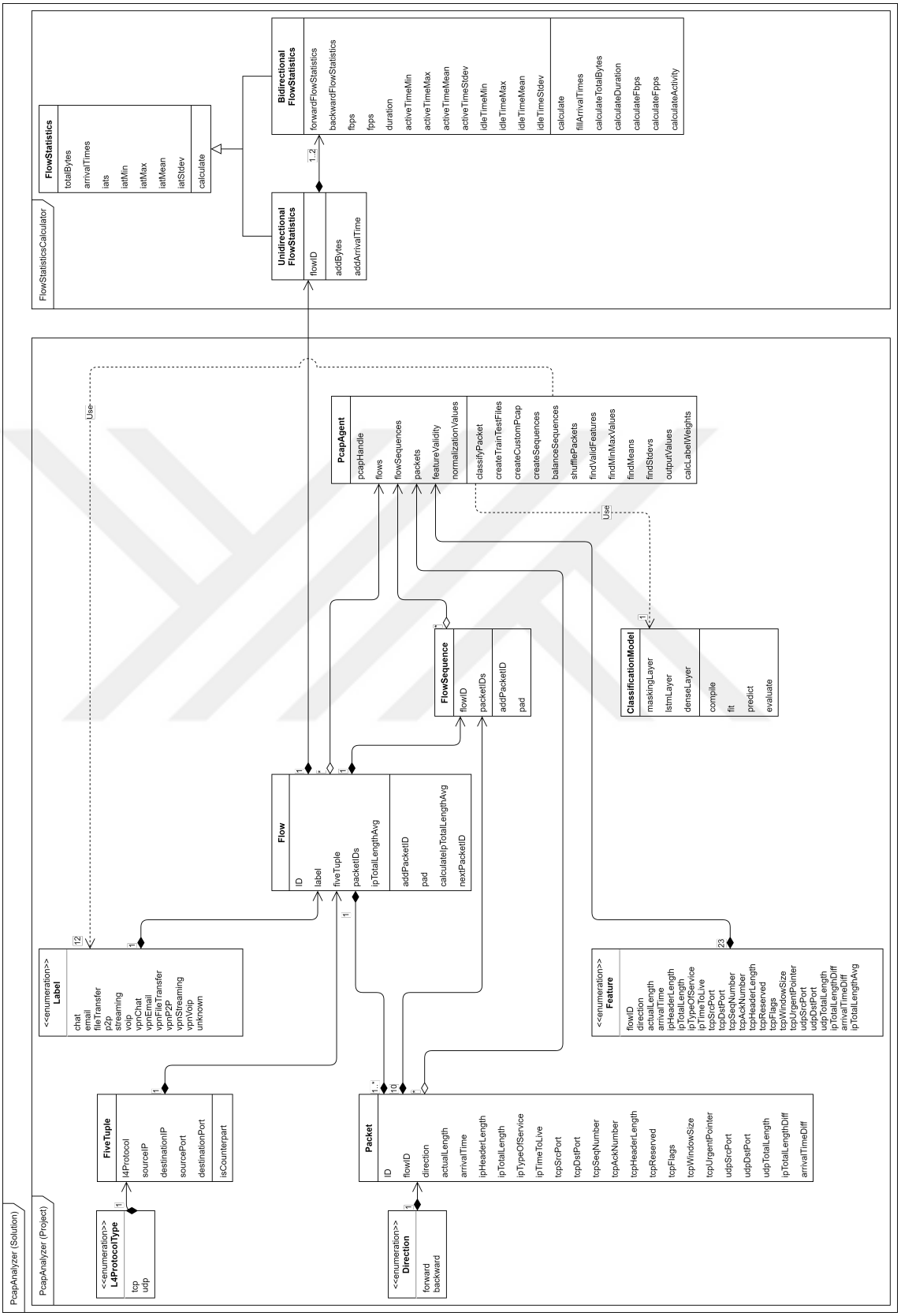

Figure 3.1. UML diagram for PcapAnalyzer solution

3.1. PCAP ANALYZER (LSTM-FS)

3.1.1. Design

In this section, the main classes that makeup PcapAnalyzer will be explained in detail. What determines a unique flow, what information we store from packets, the flow sequence concept that LSTM will use, and the LSTM model will be described.

3.1.1.1. FiveTuple

A FiveTuple is what makes a flow. Table 3.1 shows the five determinative attributes of a flow. Currently available L4ProtocolTypes are TCP and UDP. FiveTuple class has a single function as shown in Table 3.2.

Table 3.1. FiveTuple struct fields

Field	Type	Definition
l4Protocol	L4ProtocolType	Transport layer (Layer 4) protocol
sourceIP	integer	Source IP address
destinationIP	integer	Destination IP address
sourcePort	integer	Source port
destinationPort	integer	Destination port

Table 3.2. FiveTuple struct functions

Function	Parameter(s)	Return Type	Definition
isCounterpart	FiveTuple	boolean	Returns true if given five-tuple has its IP and port pairs reversed

3.1.1.2. Flow

A Flow is strictly related to a FiveTuple, a Label, and at least one non-dummy Packet. Its packets are in order by their place in the PCAP file. Currently defined Label enum fields are *chat, email, fileTransfer, p2p, streaming, voip, vpnChat, vpnEmail, vpnFileTransfer, vpnP2P, vpnStreaming, vpnVoip*, and *unknown*. Subdirectory names under the input directory must match with a Label. Tables 3.3 and 3.4 display Flow's fields and functions, respectively.

Table 3.3. Flow class fields

Field	**Type**	**Definition**
ID	integer	Unique ID
label	Label	Flow label
fiveTuple	FiveTuple	Five-tuple of flow
packetIDs	Array of integers	IDs of carried packets
ipTotalLengthAvg	float	IP total length average by now

Table 3.4. Flow class functions

Function	**Parameter(s)**	**Return Type**	**Definition**
addPacketID	integer	-	Adds given packetID to packetIDs
pad	-	-	Appends dummy packets (ID = -1) to packetIDs until it is full
calculateIpTotalLengthAvg	integer	-	Returns $\frac{(currentAvg\ N) + givenLength}{N+1}$
nextPacketID	-	integer	Returns next packetID (-1 if it is the end of flow), iterates the relevant index by 1

3.1.1.3. FlowSequence

Every FlowSequence is part of a Flow and consists of at least one non-dummy Packet 3.5. Sequences have a preset length of 10. If one has less than 10 packets, it will get padded with dummy packets 3.6. A dummy packet's all features will appear as -1. Sequences signify samples for the classification model.

Table 3.5. FlowSequence class fields

Field	Type	Definition
flowID	integer	The flow which sequence belongs to
packetIDs	integer[10]	IDs of carried packets

Table 3.6. FlowSequence class functions

Function	Parameter(s)	Definition
addPacketID	integer	Adds given packetID to packetIDs
pad	-	Appends dummy packets (ID = -1) to packetIDs until it is full

3.1.1.4. Packet

Every Packet belongs to a Flow and must have a Direction. Valid Direction enum fields are forward and backward. Packet class holds almost everything one can obtain from a network packet except the payload. Also excluding capture length, MAC addresses, IP addresses, IP Identification, fragmentation-related IP fields, and checksums since these are not used for classification. Other left-out fields are Ethernet type, IP version, and Protocol field of IP header, as these are identical for all packets (IPv4, 4, TCP/UDP, respectively). TCP members of the class are all set to 0 for UDP packets and vice versa. Most of the class members directly correspond to a feature to feed into the learning model. The class fields are shown in Table 3.7.

Table 3.7. Packet class fields

Field	Type	Definition
ID	integer	Unique ID
flowID	integer	The flow which packet belongs to
direction	Direction	Packet direction
actualLength	integer	Length on wire, frame length
arrivalTime	float	Epoch time
ipHeaderLength	integer	IP header length (IHL)
ipTypeOfService	integer	Type of Service (TOS) field of IP header
ipTotalLength	integer	IP header length + IP data length (datagram length)
ipTimeToLive	integer	Time to Live (TTL) field of IP header
tcpSrcPort	integer	TCP source port
tcpDestinationPort	integer	TCP destination port
tcpSeqNumber	integer	Sequence number field of TCP header
tcpAckNumber	integer	Acknowledgement number field of TCP header
tcpHeaderLength	integer	TCP header length (HLEN, data offset)
tcpReserved	integer	Reserved field of TCP header
tcpFlags	integer	Flags field of TCP header
tcpWindowSize	integer	Sliding window size field of TCP header
tcpUrgentPointer	integer	Urgent pointer field of TCP header
udpSrcPort	integer	UDP source port
udpDestinationPort	integer	UDP destination port
udpLength	integer	UDP header length (8 bytes) + UDP data length
ipTotalLengthDiff	integer	IP total length difference (absolute) between the previous packet
arrivalTimeDiff	float	Epoch time difference between the previous packet

3.1.1.5. PcapAgent

PcapAgent class is responsible for PCAPs and uses ClassificationModel for relevant classification procedures. It keeps Flow, FlowSequence, and Packet objects with the data gathered from the open PCAP file. Per Feature, it also holds a boolean to represent their validity; and normalization value pairs (min and max or mean and standard deviation) for feature scaling (Currently available features are flowID, direction, actualLength, arrivalTime, ipHeaderLength, ipTotalLength, ipTypeOfService, ipTimeToLive, tcpSrcPort, tcpDstPort, tcpSeqNumber, tcpAckNumber, tcpHeaderLength, tcpReserved, tcpFlags, tcpWindowSize, tcpUrgentPointer, udpSrcPort, udpDstPort, udpTotalLength, ipTotalLengthDiff, arrivalTimeDiff, and ipTotalLengthAvg. Last three are derived features.). The class uses Label enums (except ”unknown”) to map labels to flow sequence series. PcapAgent is a generic class; this way, one can store either unidirectional or bidirectional flows in it. FlowStatisticsCalculator executable uses this class to keep unidirectional flows, for example. Whereas, PcapAnalyzer executable uses it for bidirectional ones. PcapAgent fields and functions are exhibited in Table 3.8 and 3.9.

Table 3.8. PcapAgent class fields

Field	**Type**	**Definition**
pcapHandle	pcap_t	The descriptor of the open capture file (from libpcap)
flows	Array of Flows	Flows in the capture file
flowSequences	Array of FlowSequences	Flow sequences in the capture file
packets	Array of Packets	Packets in the capture file
featureValidity	Array of 23 booleans	Validity of each currently available feature
normalizationValues	Array of 23 (float, float) pairs	Normalization values for each currently available feature

Table 3.9. PcapAgent class functions

Function	**Parameter(s)**	**Return Type**	**Definition**
classifyPacket	Packet	Label	Classifies given packet
createTrainTestFiles	-	-	Outputs train and test data for classification model
createCustomPcap	-	-	Creates a custom capture file of packets
createSequences	-	Array of FlowSequences	Creates samples of flows
balanceSequences	Array of (Label, array of FlowSequences) pairs	Array of FlowSequences	Balances samples for each label
shufflePackets	-	Array of Packets	Shuffles packets retaining order in their flows
findValidFeatures	-	-	Finds and outputs valid features
findMinMaxValues	-	-	Finds minimum and maximum values for each feature
findMeans	-	-	Finds means for each feature
findStdevs	-	-	Finds standard deviations for each feature
outputValues	-	-	Outputs valid feature values required for normalization
calcLabelWeights	Array of (Label, array of FlowSequences) pairs	-	Returns $weight_i = \frac{max(s_1, s_2 \cdots s_L)}{s_i}$, where L is the number of labels, and s_i is the number of samples for i^{th} label

3.1.1.6. PcapClassifier (ClassificationModel)

PcapClassifier is a Python program that uses Tensorflow [39] and Keras [40] libraries. It builds a learning model, trains and serializes it. PcapAnalyzer imports the serialized model through an API named CppFlow [41]. ClassificationModel represents this model. Its first layer, the masking layer, helps disregard dummy values, such as values of dummy packets. The second layer, LSTM, is the heart of the model. LSTM is an RNN variant that utilizes gates to decide what to remember and forget from information learned so far. Considering it works with sequential data, series of consecutive packets is a logical candidate. Packets of a flow are never entirely independent from each other. LSTM might be able to figure out dependencies and different kinds of relations among data. Dropout is a reasonable parameter for an LSTM layer that combats with over-fitting by dropping some of the gathered information. Last is the dense layer; it has a unit for each label, and its activation function is Softmax. This function will assign probability values for each unit stating how likely is it for this label to be correct. The probabilities will add up to 1. Model compilation configuration is given in Table 3.10.

Table 3.10. Model compilation configuration

Optimizer	**Learning Rate**	**Loss Function**	**Metrics**
Adam	0.001	Sparse Categorical Crossentropy	Sparse Categorical Accuracy

As the LSTM layer of Keras library requires data to be in a 3D shape where x is the number of samples, y is timesteps or sequence length (10), and z is feature count, the input gets reshaped accordingly. It is worth noting that the LSTM layer can be structured to take varying lengths of sequences by setting the input shape parameter's first value as none. However, training in batches, which reduces training time with the help of parallelism, must still contain equally lengthed samples. This is why padding is a requirement. The batch count parameter for the LSTM layer and is set to 128. The output should contain a single label for each sequence instead of each packet; PcapAgent already handles this.

Large datasets might take a lot of memory and other resources of computers. Chunk size is an advantageous parameter for this case. It is set to 1 million, and the model's fit function runs for each chunk. This means 1 million rows are processed at a time; for example, if there are 10 million rows in total, they are processed in 10 iterations. The critical point here is to set the chunk size to such a number so that there is enough RAM to handle that many rows. Any number higher will result in an insufficient memory error. The lower the number is, the more the iterations. An optimal number can be found by experiment. Another parameter for fit is class weight. A class weight map associates each label with a coefficient. If data is imbalanced, these may help underrepresented labels to get noticed more often. PcapAgent's function to calculate label weights outputs the required coefficients. Other parameters are epoch count as 1; and validation ratio as 0.15. The validation set is different from than test set. Other than fit, Keras provides three more functions for the model: compile, predict, and evaluate. Compile builds the model based on a given configuration. Evaluate assesses the model using the provided test input and output. For adequate evaluation, test data should be completely different from train data. Finally, predict function takes a single input sequence and returns the model's prediction.

3.1.2. Usage

In this section, different usage possibilities of the PcapAnalyzer program are mentioned. PcapAnalyzer can generate training and test data from given PCAPs to feed into a learning model. Optionally, it can perform under-sampling. It shuffles the data in a way that sequential information is not lost. It extracts the necessary information according to the selected normalization technique. It can create custom PCAPs for testing purposes. The last but perhaps the most important use is offline/online classification.

3.1.2.1. Creating Train and Test Data

Among three different usages of PcapAnalyzer, the first option is to create train and test data for classification given a directory of PCAP files. Users can choose whether they want the data to be balanced per label or not. This way, any disadvantage an imbalanced dataset may bring (e.g., over-fitting) can be prevented by under-sampling. If no balancing should occur, the algorithm will output label weights for the learning model to use. To create train and test data, PcapAnalyzer executable first examines PCAP files under a given folder. The required folder structure is the same as in FlowStatisticsCalculator. Similarly, folder names get interpreted as labels. During the examination of PCAPs, each packet seen updates the PcapAgent object's packets and flows. After all PCAPs get examined, sample creation will take place, as shown in Fig. 3.2.

For each flow, a sample gets created. Since the sample length is strictly 10, flows with fewer packets get padded (Fig. 3.3). On the other hand, for flows with more, extra samples are created. To be exact, every 10 consecutive packets in a flow becomes a sample (Fig. 3.4). After samples are created, (optionally) balanced, then shuffled, 85 percent of them are training data, and the rest is testing data. It is crucial to emphasize that packets in a sequence must be in order. So, shuffling samples only means swapping ten packets in bulks.

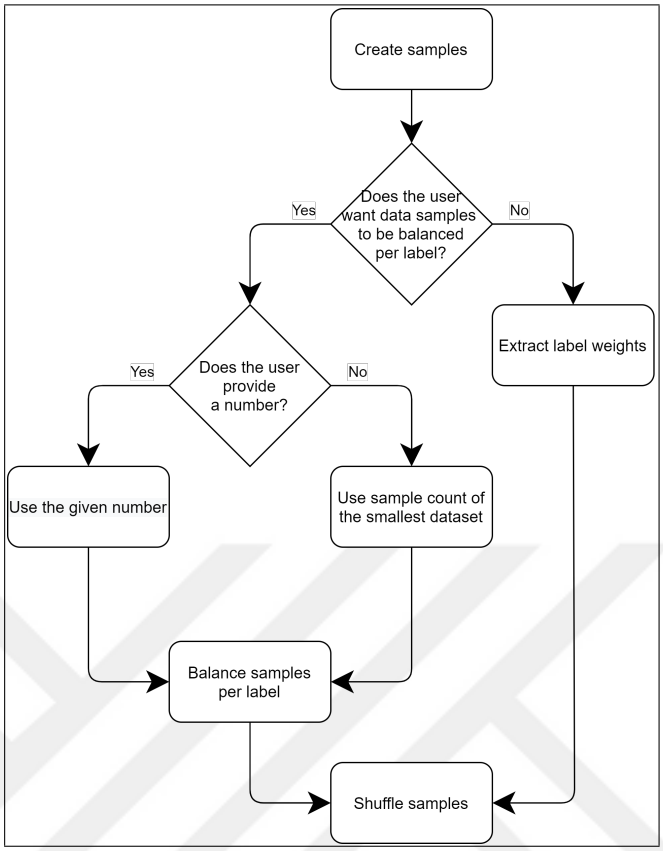

Figure 3.2. Sample creation flowchart

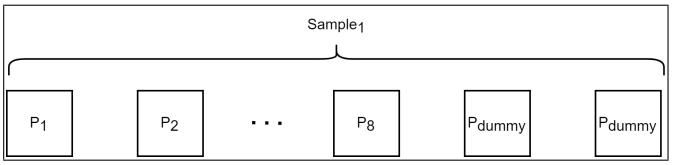

Figure 3.3. Sample creation when the number of packets is less than 10

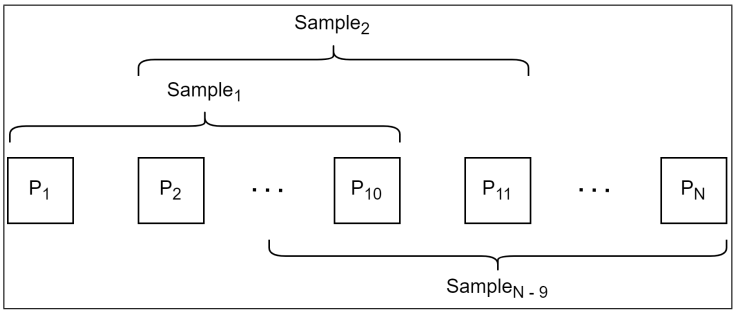

Figure 3.4. Sample creation when the number of packets is more than 10

Samples go through one more step before being written to the relevant file; feature scaling. Equations 3.1 and 3.2 shows the two options the user can decide between: min-max normalization (rescaling) or standardization (also known as Z-score normalization). In Equation 3.1, x_i is the original value for i^{th} sample, and ${x_i}'$ is the normalized value. In Equation 3.2, x_i is the original value for i^{th} sample, ${x_i}'$ is the normalized value, $\overline{x}$ is the mean, and σ_x is the standard deviation of feature x. The first option requires finding the minimum and maximum values for each feature, while the other method depends upon mean and standard deviations. Note that calculations for TCP-related features are only affected by TCP packets. Naturally, the same applies to UDP. Dummy packets are not considered. Next, invalid features are determined. If the minimum and maximum values are equal or if at least one of them is undefined, the feature is invalid. Similarly, if a feature's mean or standard deviation is 0, that feature is not valid. These features will not appear in training and testing files. PcapAnalyzer outputs training and test datasets as CSV files. Additionally, test data is extracted as a single PCAP file (dummy packets excluded); plus, a text file for labels of each packet.

$$ {x_i}' = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{3.1} $$

$$ {x_i}' = \frac{x_i - \bar{x}}{\sigma_x} \tag{3.2} $$

3.1.2.2. Creating Custom PCAPs

The second usage for the PcapAnalyzer executable is creating a custom PCAP file. Input folder structure must be the same as of first usage's. This option is handy for simulating online sniffing. One of the goals of this work is to come up with a feasible solution for online classification. To properly test this solution, an online environment or a simulation of it is required. After every PCAP under the base folder is analyzed, they get shuffled using Algorithm 3.1. Function nextPacketID ensures that packets stay in order in their flows. (Fig. 3.5) The new packet series gets dumped to a PCAP file beside a text file for each packet label.

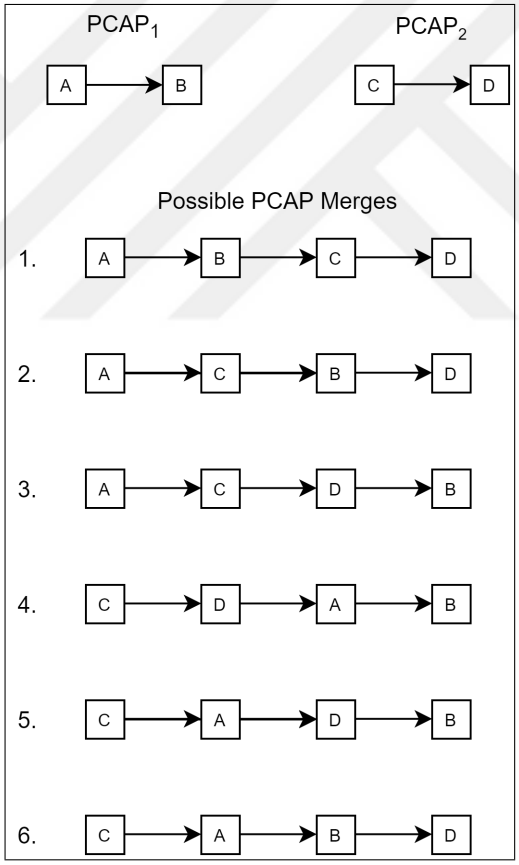

Figure 3.5. Merging PCAPs

Algorithm 3.1. Algorithm to shuffle packets retaining order in their flows

```
1: function SHUFFLEPACKETS
2:     packets ← [ ]
3:     availableFlowCount ← size(flows)
4:     while availableFlowCount > 0 do
5:         f ← random(availableFlowCount)
6:         p ← nextPacketID(f)
7:         if ID(p) ≠ −1 then
8:             Add p to packets
9:         else                                          ▷ End of flow
10:            availableFlowCount ← availableFlowCount − 1
11:            Remove f from flows
12:        end if
13:    end while
14: end function
```

3.1.2.3. Testing Offline/Online Classification

The third and final way to utilize the PcapAnalyzer executable is to test offline or simulate online traffic classification. In this case, the executable takes a single PCAP file as an input; and three text files: one for each packet's actual/expected label, another for feature validities, and valid feature values required for chosen scaling method. The executable utilizes actual packet labels for accuracy calculations; to compare expectations with predictions. Furthermore, to output a confusion matrix. Since ClassificationModel learns with a subset of features, the same must be obtained during testing, which explains the need for validities. Similarly, the same scaling technique must be applied to data with the values obtained when training. Moreover, for normalization only, any packet value less than the relevant min value will be elevated to the min, and a value more than the max will be lowered to the max. Unlike the first option, sequences are created during analysis, not after all packets are seen

(Fig. 3.6). Thus each packet can be directly sent to the classification model right after being analyzed.

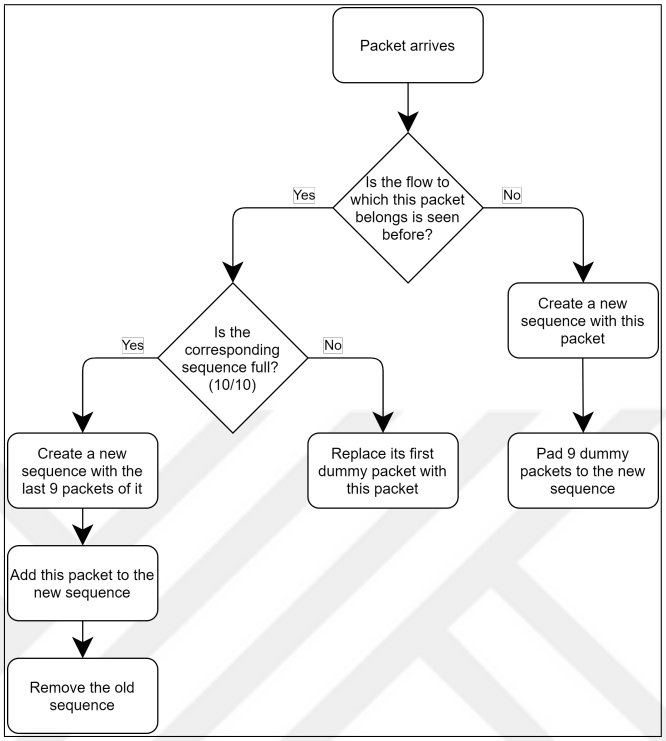

Figure 3.6. Sequence (sample) creation when testing

The program lets the user pick a number which will be the number of packets to wait to get a classification result. Since the model works with sequences of (10) packets, it makes sense to assign this number to at least 2. Any sequence having less than this number of non-dummy packets will get skipped. There is also a limit calculation for probability found by the model (Equation 3.3, L is the number of labels.). Packets with predicted results with less will be labeled as unknown. This parameter can be used to decrease the number of false positives. Misclassifications are prevented by setting a confidence limit; any prediction with lower probability will be classified as unknown, instead of accepting the predicted label no matter how low the probability is. For instance, for 2 labels ($L = 2$) we required the model to be more than 50 percent ($\frac{1}{L}$) confident with its prediction. Thus 1.5 is selected as the coefficient.

$$probabilityLimit = \frac{1}{L}1.5 \tag{3.3}$$

3.1.2.4. Thread Utilization

Simulation of online traffic classification has two key factors: mixed data and packet sender-receiver threads. Given multiple PCAP files, the PcapAnalyzer executable's second option merges them into one PCAP by rearranging packets in a way that their position in corresponding flows does not change. This mixed data is more representative of an online setting than data in which flow sequences come in sizes of 10 one after another. Option one also gives out a PCAP file, similar to the one generated for offline testing with Python. The differences are rearranged packets (to simulate online behavior better); and no dummy packets. Two threads run throughout the program, one being the sender and the other is the receiver. The threads work with a shared variable which is a fixed-size buffer where packets are stored. The executable uses lock mechanisms to prevent race conditions. The sender's job is to fill the buffer until there is no packet left. It goes back to the beginning of the buffer if the buffer is full and overwrites the old content. It also lets the reader know where to start reading and how many packets to read. This number strictly cannot be greater than the buffer size since this implies a packet drop. After every sent packet the sender sleeps for 100 nanoseconds which is the interpacket gap for a link speed of 1 Gigabit/second. An interpacket gap defines the minimum await time between two packets. On the other hand, the reader waits until the buffer is non-empty. When it can acquire the lock, it copies the relevant part of the buffer pointed by the sender. It then forwards the copied packets for classification. Thread organization helps answer questions such as how many packets can be classified in a limited amount of time; how this number is related to buffer size, and how much time it takes to process a single packet.

Although sender-receiver threads synchronize well, it is nearly impossible to measure the time elapsed for a single thread on a Windows machine. This study is only interested in the reader's performance, how fast it can process a packet and how many packets it can process

in a time frame. For this reason, to remove the sender from the equation, a distribution is generated of sent packet amounts from different program runs. The type of distribution (e.g., normal, exponential) is decided by examining these amounts, and the related calculations are made (e.g., mean and standard deviation for normal, rate parameter for exponential). The runs are done with all available cores (12). Random packet amounts are generated using the distribution until no packet is left. Up to the number of available cores, for each random amount, a reader will be spawned. For optimization, each thread will be assigned to a specific core. Since the external library class to represent the model may not be thread-safe, each reader has its copy of the model.

3.2. FLOW STATISTICS CALCULATOR (ML-SF)

3.2.1. Design

In this section, the classes of the FlowStatisticsCalculator project will be explained. Functions that calculate 23 time-based statistical features will be presented.

3.2.1.1. FlowStatistics

FlowStatistics is a base class to use as a blueprint to keep relevant statistics. Its fields are shown in Table 3.11. It has a single function which can be seen from Table 3.12.

Table 3.11. FlowStatistics class fields

Field	Type	Definition
totalBytes	integer	Summation of packet lengths
arrivalTimes	Array of N floats	Ordered series of epoch times of N packets (duplicate keys are permitted)
iats	Array of N-1 floats	Series of time differences between two consecutive packets, inter-arrival times
iatMin	float	Minimum inter-arrival time
iatMax	float	Maximum inter-arrival time
iatMean	float	Mean of inter-arrival times
iatStdev	float	Standard deviation of inter-arrival times

Table 3.12. FlowStatistics class functions

Function	Definition
calculate	Calculates iats, iatMin, iatMax, iatMean, and iatStdev

3.2.1.2. UnidirectionalFlowStatistics

This is a derived class of FlowStatistics. It has no meaning without a Flow object 3.13. Functions it has are exhibited in Table 3.14.

Table 3.13. UnidirectionalFlowStatistics class fields

Field	Type	Definition
flowID	integer	The flow which statistics belongs to

Table 3.14. UnidirectionalFlowStatistics class functions

Function	Parameter(s)	Definition
addBytes	integer	Adds given length to totalBytes
addArrivalTime	float	Adds given time to arrivalTimes

3.2.1.3. BidirectionalFlowStatistics

This is a derived class of FlowStatistics. It has no meaning without an UnidirectionalFlowStatistics object, whereas it can carry at most two 3.15. ”iats” field of these objects correspond to what Draper-Gil et al. [1] referred to as ”fiat” and ”biat”. Naturally, the field ”iats” of this class is equivalent to ”flowiat”. The required UnidirectionalFlowStatistics represents a forward flow, and the optional one, a backward flow. For example, the forward flow may consist of client-to-server packets, while the other contains server-to-client packets. BidirectionalFlowStatistics functions are shown in Table 3.16.

Table 3.15. BidirectionalFlowStatistics class fields

Field	Type	Definition
forwardFlowStatistics	UnidirectionalFlowStatistics	Statistics of the forward flow
backwardFlowStatistics	UnidirectionalFlowStatistics	Statistics of the backward flow
duration	float	Duration of flow
fbps	float	Flow bytes per second
fpps	float	Flow packets per second
activeTimeMin	float	Minimum active time
activeTimeMax	float	Maximum active time
activeTimeMean	float	Mean of active times
activeTimeStdev	float	Standard deviation of active times
idleTimeMin	float	Minimum idle time
idleTimeMax	float	Maximum idle time
idleTimeMean	float	Mean of idle times
idleTimeStdev	float	Standard deviation of idle times

Table 3.16. BidirectionalFlowStatistics class functions

Function	Definition
calculate	Calculates all statistics
fillArrivalTimes	Fills arrivalTimes by joining the forward and the backward flow's
calculateTotalBytes	Calculates totalBytes by adding the forward and the backward flow's
calculateDuration	Returns $duration = arrivalTimes_N - arrivalTimes_1$, where N is the number of arrivals
calculateFbps	Returns $fbps = \frac{totalBytes}{duration}$
calculateFpps	Returns $fpps = \frac{N}{duration}$
calculateActivity	Calculates active and idle times using Algorithm 3.2

Algorithm 3.2. Algorithm to calculate active and idle times

```
1:  function CALCULATEACTIVITY
2:      activeTimes ← [], idleTimes ← []
3:      activityLimit ← iatMean
4:      activeTimeSum ← 0
5:      for each iat do
6:          if iat < activityLimit then
7:              activeTimeSum ← activeTimeSum + iat
8:          else
9:              Add iat to idleTimes
10:             if activeTimeSum ≠ 0 then
11:                 Add activeTimeSum to activeTimes
12:                 activeTimeSum ← 0
13:             end if
14:         end if
15:     end for
16:     if activeTimeSum ≠ 0 then
17:         Add activeTimeSum to activeTimes
18:     end if
19: end function
```

3.2.2. Usage

FlowStatisticsCalculator executable analyzes PCAP files under subdirectories of a given directory. It takes subdirectory names as labels. It outputs an ARFF (Attribute-Relation File Format) file including calculated statistics ready to be fed into Weka [42]. The main program makes use of the PcapAgent generic class of the PcapAnalyzer project to store unidirectional flows.

TCP flows typically end with a FIN-ACK sequence between client and server; or an RST packet from the server. On the other hand, UDP flows terminate when they reach a predetermined timeout. Draper-Gil and his team [1] observed that 15 seconds is an adequate timeout. For every packet, FlowStatisticsCalculator calculates the current duration of the packet's flow and starts a new one if the timeout has been reached.

4. RESULTS AND DISCUSSION

In this chapter, the dataset used in this work and how it is utilized, test scenarios, and hardware features are described; test results for LSTM-FS (LSTM with Flow Sequences) and ML-SF (Machine Learning with Statistical Features) are given. Comparisons were made among test outcomes.

4.1. DATASET

The experiments use the ISCXVPN2016 dataset [16]. The dataset contains all encrypted packets; non-VPN, VPN, and Tor samples. Tor is ruled out from tests for the sake of simplicity. There exist six non-VPN and six VPN categories for chat, email, file transfer, P2P, streaming, VoIP. As opposed to Draper-Gil et al. [1], this work does not include the ”browsing” label since the others match better with the PCAPs. For the transport layer, TCP and UDP are supported, the program ignores ICMP and IGMP packets. PCAPs with raw IP packets, almost all VPN data, are padded with Ethernet headers. Doing this provides a way to merge multiple PCAPs with different link types.

4.2. TEST SCENARIOS

The study by Draper-Gil et al. [1] has four test scenarios: A1, A2 non-VPN, A2 VPN, and B, as summarized in Table 4.1. The same scenarios are built for the methods ML-SF and LSTM-FS to make meaningful comparisons. In the first part of Scenario A (A1), the researchers split the dataset into two labels, non-VPN and VPN, to feed into Weka (Fig. 4.1). In the second part of Scenario A (A2 non-VPN and A2 VPN), they divide each label’s data into six categories in itself and handle both separately (Fig. 4.2 and Fig. 4.3). In Scenario B, this study uses 12 labels (six non-VPN and six VPN categories) to perform classification in one step (Fig. 4.4).

All experiment scenarios ran both offline and online for LSTM-FS. ML-SF method is tried both flow-based and packet-based.

Table 4.1. Test scenarios

Scenario	Labels
A1	non-VPN, VPN
A2 non-VPN	chat, email, file transfer, P2P, streaming, VoIP
A2 VPN	VPN chat, VPN email, VPN file transfer, VPN P2P, VPN streaming, VPN VoIP
B	chat, email, file transfer, P2P, streaming, VoIP, VPN chat, VPN email, VPN file transfer, VPN P2P, VPN streaming, VPN VoIP

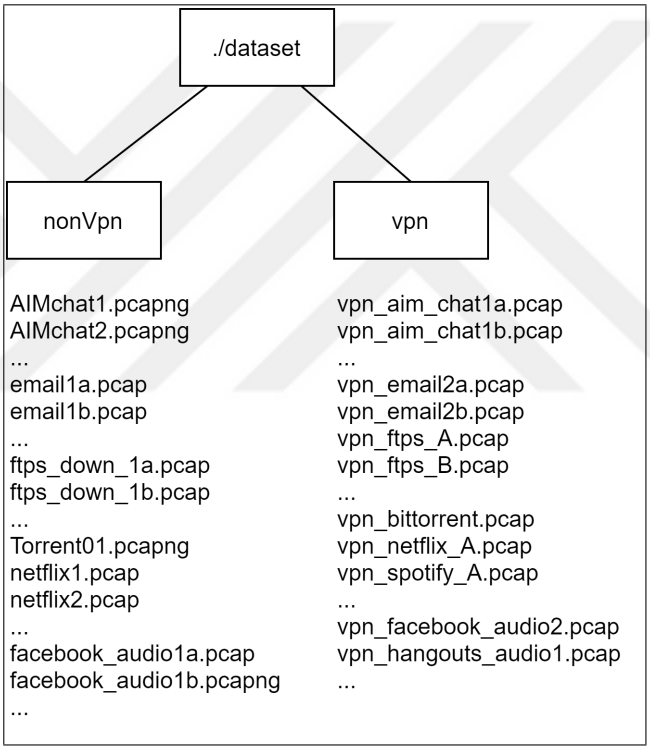

Figure 4.1. Input directory structure for Scenario A1

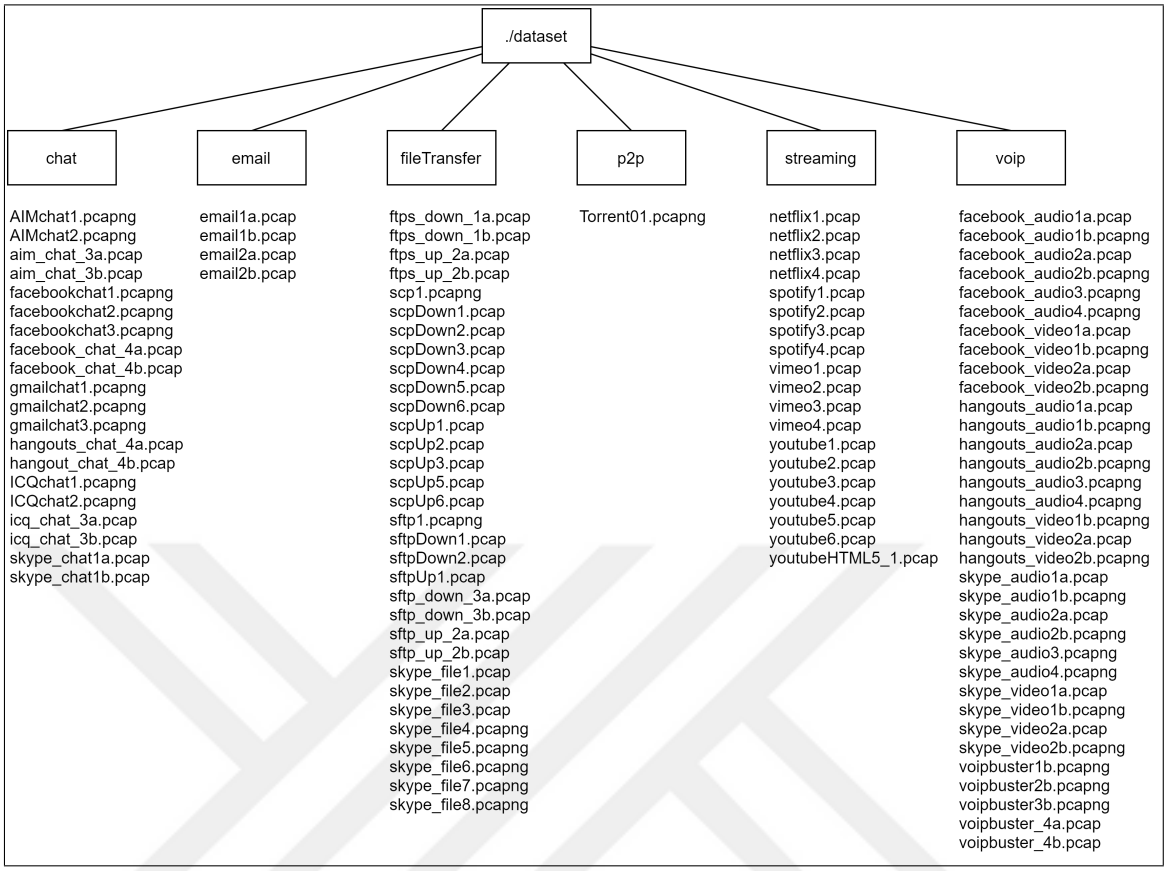

Figure 4.2. Input directory structure for Scenario A2 non-VPN

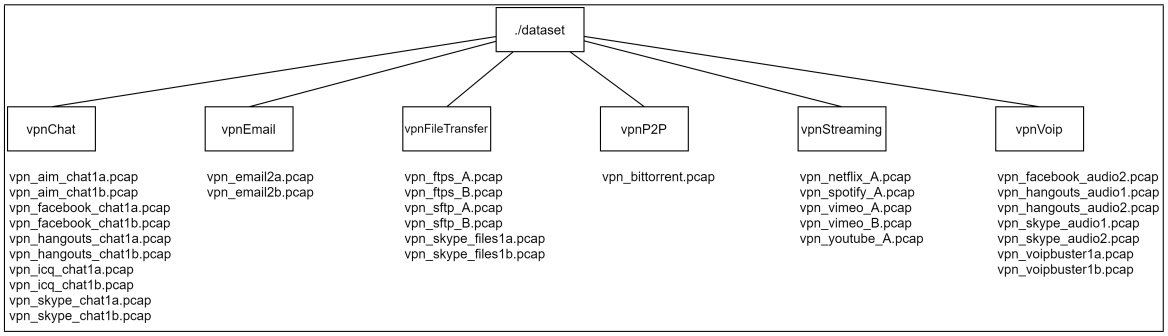

Figure 4.3. Input directory structure for Scenario A2 VPN

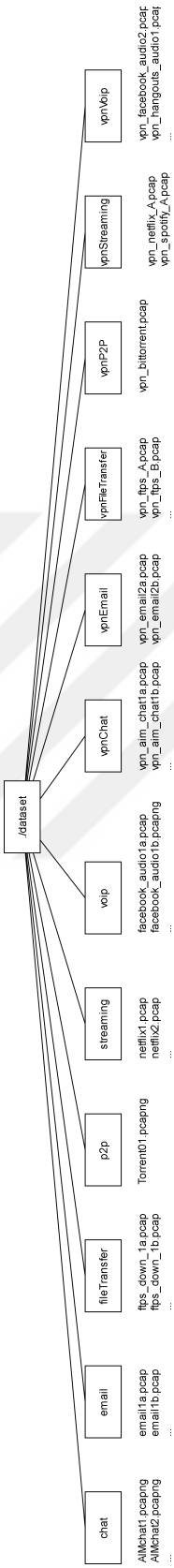

Figure 4.4. Input directory structure for Scenario B

4.3. EXPERIMENTAL SETUP

4.3.1. Hardware Specifications

All tests are executed on a commercial PC with Intel Core i7-8750H CPU @ 2.20GHz and 16 GB RAM. The CPU has 12 cores.

4.3.2. Data Sampling

4.3.2.1. LSTM-FS

For LSTM-FS, there is no difference between offline and online data; LSTM-FS calls each sequence of 10 packets a sample.

Table 4.2. LSTM Scenario A1 sample distribution before sampling

Label	Samples
nonVpn	20,998,396
vpn	4,796,577

Table 4.3. LSTM Scenario A2 non-VPN sample distribution before sampling

Label	Number of Samples
chat	91,509
email	18,329
fileTransfer	10,736,535
p2p	105,031
streaming	729,343
voip	7,459,544

Table 4.4. LSTM Scenario A2 VPN sample distribution before sampling

Label	Number of Samples
vpnChat	80,497
vpnEmail	20,132
vpnFileTransfer	372,354
vpnP2P	419,646
vpnStreaming	1,510,282
vpnVoip	2,393,401

Since it takes too long to process millions of data (See Tables 4.2, 4.3, 4.4), LSTM-FS creates sub-datasets. For every scenario, 10,000 data is chosen randomly per label (Fig.4.5), in a way that packet order in sequences is preserved. For its tests, it repeats this process five times to generate five models.

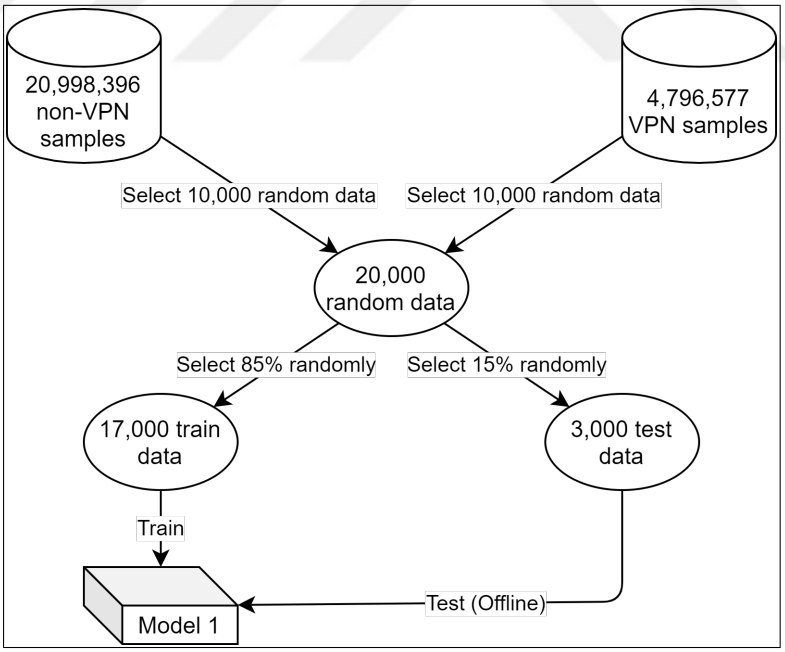

Figure 4.5. LSTM-FS model generation example (Scenario A1)

4.3.2.2. ML-SF

For flow-based classification, ML-SF calculates statistics per flow after reading all packets. Then every statistic becomes a sample. In other words, the number of flows is equal to the number of data samples. There is no data sampling required for this case since tests finish in a reasonable time, plus Weka can handle the amount of data with no difficulty. For packet-based classification, ML-SF calculates statistics after each packet arrival. The number of samples is the number of packets. Updating statistics after each network packet then writing to a file (to feed Weka) takes a considerable amount of time and memory, especially when there are millions. Besides, resulting files are significantly big for Weka to build, train and test a model. For example, when no sampling is done, it takes approximately 36 hours to generate training data for Scenario A2 non-VPN, which results in a 7.92 GB ARFF file. Then, during training, Weka aborts with an out-of-memory error even if the maximum heap size for Weka is as much as possible. Since this scenario's dataset is much smaller than A1's and B's, it is clear that these scenarios cannot complete too, with the current setup. Only for Scenario A2 VPN successful training is doable by increasing the heap size to 8 GB.

Fig. 4.6 shows how ML-SF samples data to make packet-based classification possible with the available configuration. ML-SF chooses 10,000 samples (packets) for each label, but this cannot be random; it should only use consecutive packets since its features depend on inter-arrival times. Since randomness is involved, the sampling process repeats five times to generate five models. Equation 4.1 presents the formula to calculate the number of packets to skip before extracting 10,000 samples for the current label; s_i is the number of packets to skip, and c_i is the sample count for i^{th} label; $j = 0$ for $Model_1$, ... $j = 4$ for $Model_5$ thus $S(j) = 5$.

$$s_i = (\frac{c_i - 10000}{S(j) - 1}) * j \tag{4.1}$$

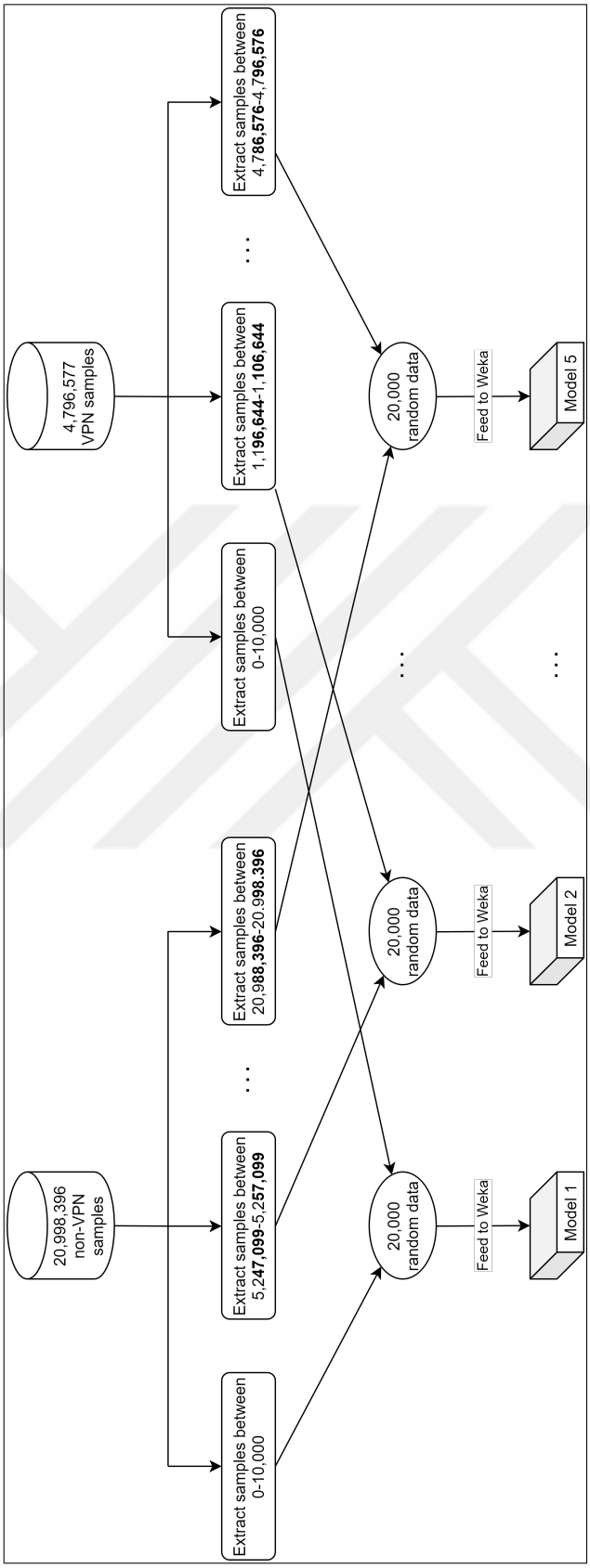

Figure 4.6. ML-SF model generation example

Table 4.5. ML-SF Scenario A1 sample distributions

Label	**Samples**	
	Flow-Based	**Packet-Based (before sampling)**
nonVpn	194,420	21,294,666
vpn	29,760	4,831,046

Table 4.6. ML-SF Scenario A2 non-VPN sample distributions

Label	**Samples**	
	Flow-Based	**Packet-Based (before sampling)**
chat	10,124	104,611
email	3,307	26,352
fileTransfer	39,932	10,795,681
p2p	903	108,433
streaming	4,425	737,866
voip	81,527	7,686,775

Table 4.7. ML-SF Scenario A2 VPN sample distributions

Label	**Samples**	
	Flow-Based	**Packet-Based (before sampling)**
vpnChat	5,385	86,040
vpnEmail	468	21,558
vpnFileTransfer	3,180	376,226
vpnP2P	524	422,018
vpnStreaming	2,929	1,514,095
vpnVoip	17,209	2,410,093

The sample distributions are given in Table 4.5, 4.6 and 4.7; for Scenario B it is omitted since it is simply the combination of Scenario A-2 tables. The distribution for packet-based classification after sampling is also nonapparent since each label has simply 10,000 samples.

4.3.3. Parameters

All LSTM-FS models use the same 17 features (out of 23 available). Four of these features, namely, flowID, ipTotalLengthDiff, arrivalTimeDiff, ipTotalLengthAvg, are excluded beforehand due to the unnecessary complexity they bring. Features ipHeaderLength and tcpUrgentPointer are discarded by the algorithm since they are the same for all chosen samples. The feature scaling method is normalization. Dropout is 0.1. The sequence length is 10. 75 percent of the data is used to train, 15 percent to validate, and 10 percent to test. Class weight optional parameter is unused since there already exists an equal amount of data per label. Chunk size is 1 million, the batch count is 128. Epoch count is 1. For scenarios A2 non-VPN and B, Adam learning rate was changed to 0.01 from 0.001. (This had an impact on accuracy around 3 percent.) For online classification, 1 sender and 11 reader threads test each model first. The goal is to decide on an appropriate distribution type for sent packet amounts to replace the sender thread. The packet buffer is made large enough to prevent packet drops. 10 tests run to get average accuracy and time results. There is no restriction on how many packets LSTM-FS should see before classification; every sample immediately enters the classification model. A sample is classified as unknown if the best probability is under a limit which a formula given in Chapter 3 calculates. It is important to note that different runs yield different results since each thread has its PcapAgent object, so each has its flows and sequences.

For ML-SF, in both flow-based and packet-based classification, the flow timeout parameter is 15 seconds since Draper-Gil et al. [1] got the best results with it. Meaning that if a flow takes longer than 15 seconds, a new flow with separate statistics gets generated where each statistic is a sample. Training features include the minimum, maximum, mean, and standard deviation of active-idle time, inter-arrival times for packets sent in each direction (forward, backward) and both directions; flow packets per second, flow bytes per second. In Weka, OneR (One Rule) and C4.5 (J48) algorithms trained separate models with given data, then

the cross-validation method with 10 folds tested the models. OneR results revealed whether the classification problems are simple or not; J48 has the highest accuracies in [1]. Since there is no difference between multiple runs of the algorithms (there is no randomness; the algorithms are deterministic), tests reran only to get an average of elapsed times.

4.4. TEST RESULTS

Average accuracy results, average elapsed times, and confusion matrices are shared for all test scenarios. Time values presented are in seconds; averaged results denote that the related test ran five times; unless specified otherwise. Accuracy column corresponds to correctly classified instances percentage; precision, recall, and F-measure are weighted averages.

4.4.1. LSTM-FS

Offline and online experiment results for LSTM-FS are given in this section.

4.4.1.1. Offline Classification

Table 4.8 exhibits the average accuracy results of offline classification with LSTM-FS for all scenarios. Table 4.9 shows the average elapsed times. Table 4.10, 4.11, 4.12, and 4.13 displays the average confusion matrices for each scenario.

Table 4.8. LSTM-FS (Offline) average accuracy results

Scenario	Accuracy	Precision	Recall	F-Measure
A1	99.818	0.9982	0.9982	0.9982
A2 non-VPN	97.988	0.9804	0.9799	0.9799
A2 VPN	96.828	0.9691	0.9683	0.9683
B	96.43	0.9658	0.9643	0.9644

Table 4.9. LSTM-FS (Offline) average elapsed times

Scenario	Data Generation	Training	Testing
A1	562.73	12.68	2.39
A2 non-VPN	291.39	25.96	3.54
A2 VPN	26.62	28.08	3.73
B	432.17	68.29	7.36

Table 4.10. LSTM-FS (Offline) Scenario A1 average confusion matrix

Actual \ Predicted	VPN	non-VPN
VPN	1,513.2	5
non-VPN	0.4	1,481.4

Table 4.11. LSTM-FS (Offline) Scenario A2 non-VPN average confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip
chat	1,411.8	21.6	13.2	0.2	4	32.2
email	10.8	1,449.8	0.6	0	0	0
fileTransfer	1.4	4.8	1,502.2	0	2.2	22.2
p2p	2	0	0	1,462.4	12	39.4
streaming	0	0	0	1.6	1,492.4	0.2
voip	0.6	1	0	10.4	0.4	1,500.6

Table 4.12. LSTM-FS (Offline) Scenario A2 VPN average confusion matrix

Actual \ Predicted	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
vpnChat	1,400.2	3.4	5.2	111.2	0.8	0
vpnEmail	4.6	1,458.8	14.4	4	0	0
vpnFileTransfer	59.8	6.6	1,394	24.2	4	0.8
vpnP2P	24.8	6.4	11.2	1,463.6	0	0
vpnStreaming	0	0	4	0	1,501.4	0
vpnVoip	0	0	0	0	0	1,496.6

Table 4.13. LSTM-FS (Offline) Scenario B average confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
chat	1,437.6	15.8	0.2	0	0.6	26.8	0	0	0	0	0	0
email	4.4	1,495	0	0	0	0	2.4	0	5	0.2	0	0
fileTransfer	2.4	1.4	1,473.6	0	0.8	18.6	0	0	0.6	0	0	0.6
p2p	1.4	0	0	1,442	12.8	40.2	0	0	0	0	0	0.4
streaming	0.2	0	0	0.6	1,509.8	0	0	0	0	0	0	0
voip	5	0.2	0	6.2	0.4	1,497.2	0	0	0	0	0	0
vpnChat	0	1.8	0	0	0	0	1,353.8	19.4	20.6	71.2	8.4	0
vpnEmail	0	4.6	0	0	0	0	2.2	1,451.2	11.8	20.8	0	0
vpnFileTransfer	0.2	19.2	1	0	0	0	12.6	10.8	1,398.8	12	58.4	2
vpnP2P	0	4.6	0	0	0	0	19.6	2.2	16.2	1,460.4	0.4	0
vpnStreaming	0	0.4	0	0	0	0	0	0	162.2	0.2	1,360.6	0
vpnVoip	3.6	0	0	3	4	0	0	0	1.6	0	0	1,477.8

4.4.1.2. Online Classification

After five runs for the first model, as can be seen from Fig. 4.7 that the sender tends to send a low number of packets rather than a high number. In other words, the sender rarely holds the CPU for long and sends a high number of packets at once; on the contrary, it releases the CPU quite often, resulting in a lower number of piled-up packets. The mean and standard deviation is 81.3062 and 497.534, respectively. The exponential distribution fits this data. The rate parameter (λ) is 0.0122 according to the Equation 4.2. The maximum amount of sent packets is 7,970. However, the packet buffer size is decided as 25,000 since further experiments showed that sometimes each reader gets a few packets, then the remaining packets are received by the one who finishes reading first.

$$\lambda = \frac{1}{\mu} \tag{4.2}$$

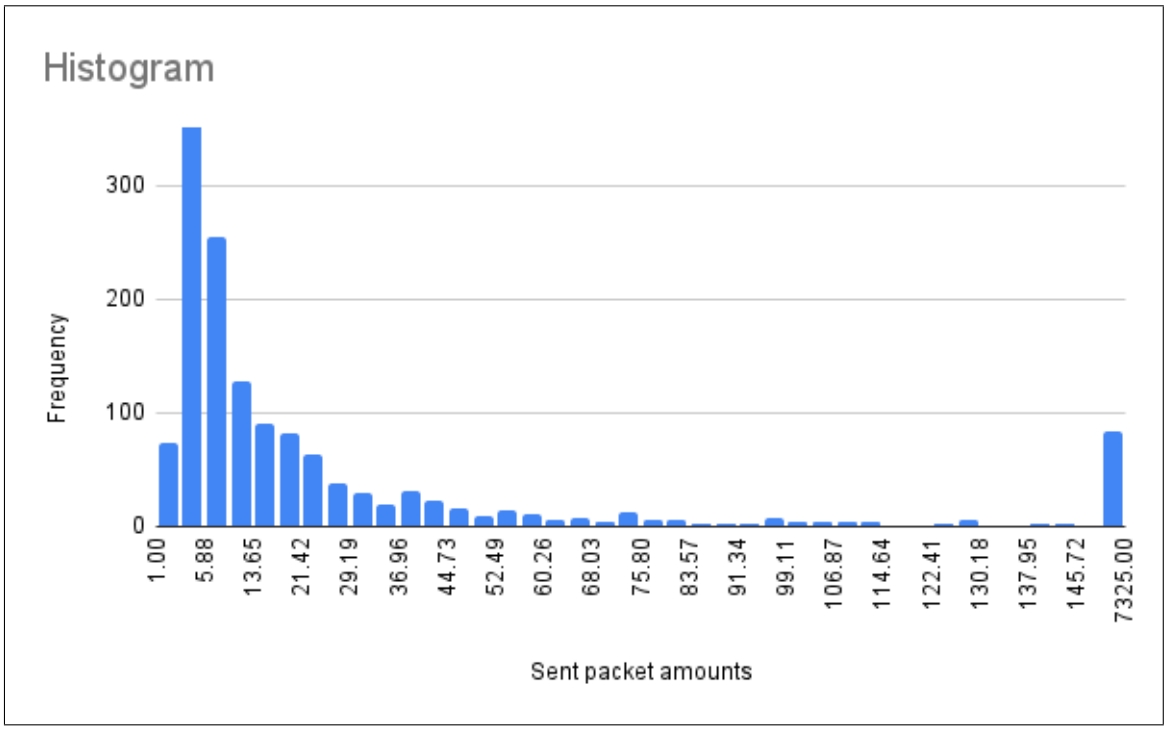

Figure 4.7. LSTM-FS (Online) Scenario A1 model 1 sent packets histogram

Table 4.14 exhibits the average accuracy results of online classification with LSTM-FS for all scenarios. Table 4.15 shows the average elapsed times. Table 4.16, 4.17, 4.18, and 4.19 displays the average confusion matrices for each scenario.

Table 4.14. LSTM-FS (Online) accuracy results

Scenario	**Accuracy**	**Precision**	**Recall**	**F-Measure**	**Unknown (percent)**
A1	99.8554	0.9986	0.9986	0.9986	6.93
A2 non-VPN	94.8119	0.9551	0.9515	0.9506	0.65
A2 VPN	83.8309	0.8477	0.8346	0.8319	5.6
B	88.3211	0.9098	0.8849	0.8874	0.1

Table 4.15. LSTM-FS (Online) average elapsed times

Scenario	**Testing**	**Packet/Second**	**Packet Processing Overhead (ms)**
A1	14.45	1,856.99	0.5398
A2 non-VPN	60.81	1,355.75	0.7477
A2 VPN	35.73	2,466.01	0.4058
B	116.89	1,529.62	0.6786

Table 4.16. LSTM-FS (Online) Scenario A1 average confusion matrix

Actual \ Predicted	VPN	non-VPN
VPN	12,450.1	36.2
non-VPN	0	12,442.58

Table 4.17. LSTM-FS (Online) Scenario A2 non-VPN average confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip
chat	13,512.18	197.14	113.66	18.92	423.12	84.48
email	119.78	12,114.98	42.62	0	49.46	18.7
fileTransfer	2.44	10.88	10,876.76	0	24.34	46.2
p2p	63.26	0.12	0	12,852.54	386.42	1,646.08
streaming	0.02	0.06	20.14	6.32	14,032.86	63.54
voip	15.78	0.88	0	403	433.4	13,170.5

Table 4.18. LSTM-FS (Online) Scenario A2 VPN average confusion matrix

Actual \ Predicted	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
vpnChat	6,668.1	4,782.36	480.62	1,124.74	0.82	0
vpnEmail	1,429.42	11,408.48	203.62	251.44	0	0
vpnFileTransfer	642.76	77	12,417.64	158.44	28.4	0.14
vpnP2P	444.6	2,754.3	295.02	11,012.84	0.52	0
vpnStreaming	1.9	0	156.04	405.96	14,376.52	0
vpnVoip	0	0	201.34	0	1.58	13,786.98

Table 4.19. LSTM-FS (Online) Scenario B average confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
chat	13,429.5	108.44	25.44	1.66	704.92	72.72	0.44	0	106	0.14	0	7.46
email	37.02	10,617.04	3.06	0	2.42	4.18	38.84	10.7	2,177.52	1.24	0	0
fileTransfer	9.64	0.72	11,996.96	0	34.16	41.08	0.26	0	46.82	0	0	0.96
p2p	112.48	0	0.08	11,588.88	746.82	2,396.58	0	0	0	0	0	3.56
streaming	0.32	0	0	2.36	14,188.9	63.82	0	0	0	0	0	44.2
voip	43.52	0.32	0.28	135.36	371.34	13,509.24	0	0	0	0	0	0.14
vpnChat	0.32	158.4	0.02	0	5.92	0	10,157.3	364.44	3,453.02	376.2	221.16	0
vpnEmail	0.06	19.54	0.08	0	0.06	0	135.04	13,296.28	1,062.76	265.92	3.3	0
vpnFileTransfer	0.64	87.54	15.08	0.42	17.42	0	443.12	174.06	13,744.62	23.42	410.6	14.46
vpnP2P	0	15.74	0	0	0.9	0	211.52	1,350.42	324.52	12,989.86	1.98	0
vpnStreaming	0.42	2.76	0	0	15.14	0	243.72	41.2	2,292.6	53.36	12,517.58	0.06
vpnVoip	19.68	30.1	0.04	17.9	811.5	0.6	0	0	44.88	0	0	13,951.3

4.4.2. ML-SF

Flow-based and packet-based experiment results for ML-SF are given in this section.

4.4.2.1. Flow-Based Classification

In [1], the researchers observed at most around 90-95 percentages for Scenario A1. Whereas here, almost every result is near 100 percent (Table 4.20). Different labeling may be the best candidate to explain why. For example, in our study, the browsing label was not used because other labels were thought to better express the data. Table 4.21 displays average elapsed times. Examining the graphs generated by Weka, none of the features seem as if they can single-handedly manage classification (Fig. 4.8). Using Weka's OneR classifier reveals that flowiatMin is the closest feature to do so. Even OneR performs great, even though it is a basic algorithm. Both algorithms seem to be keen on mistaking VPN by non-VPN, which may be related to data imbalance between the two labels that stands out.

Table 4.20. ML (Flow) accuracy results

Scenario	**Accuracy**		**Precision**		**Recall**		**F-Measure**	
	OneR	**J48**	**OneR**	**J48**	**OneR**	**J48**	**OneR**	**J48**
A1	98.9419	99.8269	0.989	0.998	0.989	0.998	0.989	0.998
A2 non-VPN	99.6391	99.8609	0.996	0.999	0.996	0.999	0.996	0.999
A2 VPN	100	99.9798	1	1	1	1	1	1
B	98.3662	99.6545	0.984	0.997	0.984	0.997	0.983	0.997

Table 4.21. ML (Flow) average elapsed times

Scenario	**Data Generation**	**Training**		**Testing**	
		OneR	**J48**	**OneR**	**J48**
A1	111.42	2.15	13.2	24.45	128.6
A2 non-VPN	84.82	0.76	9.22	8.24	83.78
A2 VPN	10.5	0.14	0.25	1.06	2.35
B	112.08	1.14	10.33	14.46	108.67

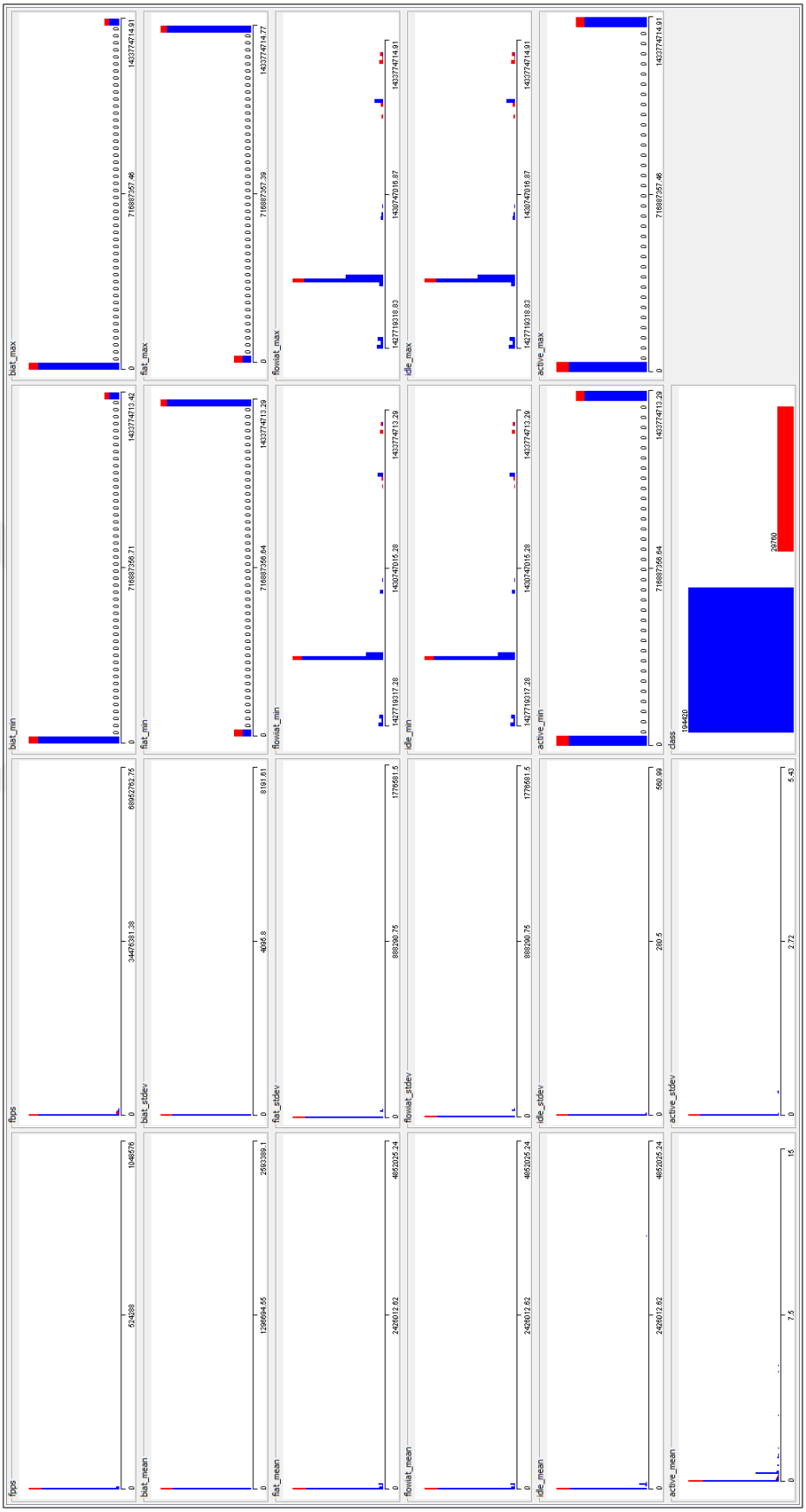

Figure 4.8. ML (Flow) Scenario A1 features visualized

Table 4.22 shows the confusion matrix for Scenario A1 when OneR algorithm is used, whereas Table 4.23 displays J48 results.

Table 4.22. ML (Flow) Scenario A1 OneR confusion matrix

Actual \ Predicted	**VPN**	**non-VPN**
VPN	27,670	2,090
non-VPN	282	194,138

Table 4.23. ML (Flow) Scenario A1 J48 confusion matrix

Actual \ Predicted	**VPN**	**non-VPN**
VPN	29,537	223
non-VPN	165	194,255

OneR chooses feature flowiatMin for Scenario A2 non-VPN. J48 is slightly better than OneR (See Table 4.24, 4.25). Observing the confusion matrix, the model tends to classify flows as VoIP, and this may again be because that label is the majority of the data.

Table 4.24. ML (Flow) Scenario A2 non-VPN OneR confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip
chat	9,758	0	0	0	0	366
email	0	3,307	0	0	0	0
fileTransfer	0	0	39,931	0	0	1
p2p	0	0	0	902	0	1
streaming	0	0	0	0	4,334	91
voip	37	0	4	0	6	81,480

Table 4.25. ML (Flow) Scenario A2 non-VPN J48 confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip
chat	9,999	1	5	0	0	119
email	1	3,306	0	0	0	0
fileTransfer	8	1	39,919	0	1	3
p2p	0	0	0	903	0	0
streaming	0	0	1	1	4,412	11
voip	26	0	2	0	15	81,484

Features flowiatMin, flowiatMax, idleMin, and idleMax drew almost identical graphs for Scenario A2 VPN. Looking at Fig. 4.9, it is natural to speculate that these features can differentiate between classes doing only a few conditional checks. Such that, OneR results in 100 percent accuracy with any of them. On the other hand, having other features seem to confuse J48; this is the only case where J48 yields (slightly) lower accuracy than OneR (See Table 4.26). Perfect accuracy seems suspicious. With the four features removed, OneR's accuracy decreases to 81.0945 percent, while J48 results 97.737 percent. Labels vpnEmail and vpnP2P get dramatically affected by the lack of the four features (Table 4.27, 4.28)

For Scenario B, both algorithms are prone to confuse voip with vpnVoip (Table 4.29, 4.30).

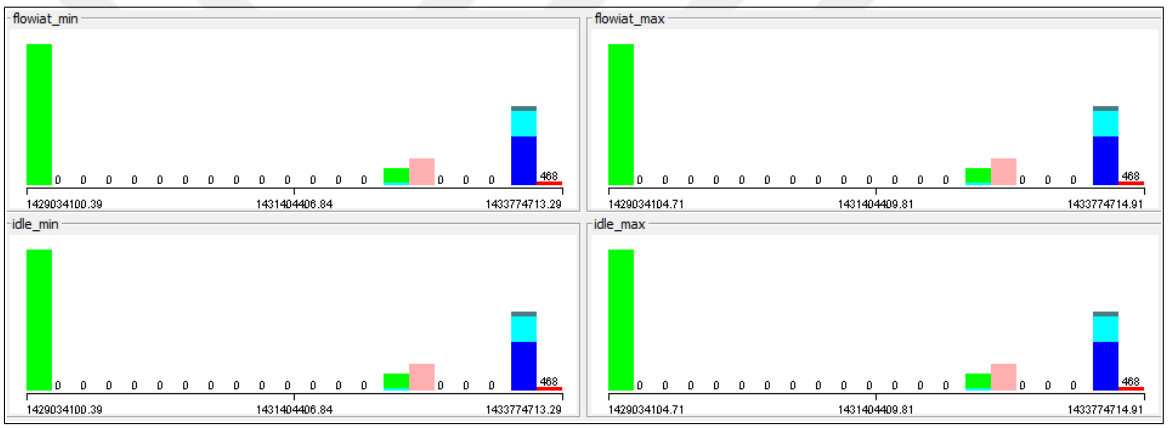

Figure 4.9. ML (Flow) Scenario A2 VPN four features visualized

Table 4.26. ML (Flow) Scenario A2 VPN J48 confusion matrix

Actual \ Predicted	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
vpnChat	5,384	0	0	1	0	0
vpnEmail	0	468	0	0	0	0
vpnFileTransfer	0	1	3,178	0	1	0
vpnP2P	0	0	1	523	0	0
vpnStreaming	1	0	0	0	2,928	0
vpnVoip	0	0	1	0	0	17,208

Table 4.27. ML (Flow) Scenario A2 VPN OneR matrix, 4 features removed

Actual \ Predicted	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
vpnChat	4,283	39	442	17	73	531
vpnEmail	164	63	116	4	20	101
vpnFileTransfer	386	49	1,705	24	300	716
vpnP2P	100	16	173	84	43	108
vpnStreaming	100	17	272	25	1,973	542
vpnVoip	295	15	532	40	354	15,973

Table 4.28. ML (Flow) Scenario A2 VPN J48 matrix, 4 features removed

Actual \ Predicted	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
vpnChat	5,265	34	24	12	15	35
vpnEmail	56	401	1	5	1	4
vpnFileTransfer	49	5	2,992	19	77	38
vpnP2P	13	2	13	472	4	20
vpnStreaming	9	0	16	1	2,855	48
vpnVoip	36	4	54	14	63	17,038

Table 4.29. ML (Flow) Scenario B OneR confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
chat	9,750	0	0	0	0	374	0	0	0	0	0	0
email	0	3,307	0	0	0	0	0	0	0	0	0	0
fileTransfer	0	0	39,930	0	0	2	0	0	0	0	0	0
p2p	0	0	0	903	0	0	0	0	0	0	0	0
streaming	0	0	0	0	4,334	91	0	0	0	0	0	0
voip	29	0	3	0	1	81,150	0	0	0	0	0	344
vpnChat	0	0	0	0	0	0	5,385	0	0	0	0	0
vpnEmail	0	0	0	0	0	0	0	468	0	0	0	0
vpnFileTransfer	0	0	0	0	0	0	0	0	3,180	0	0	0
vpnP2P	0	0	0	0	0	0	0	0	0	524	0	0
vpnStreaming	0	0	0	0	0	0	0	0	0	0	2,929	0
vpnVoip	0	0	0	0	0	1,932	0	0	0	0	0	15,277

Table 4.30. ML (Flow) Scenario B J48 confusion matrix

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
chat	10,002	0	6	0	0	89	1	0	0	0	0	26
email	1	3,305	0	0	0	0	0	1	0	0	0	0
fileTransfer	4	0	39,922	0	1	2	0	0	0	0	0	3
p2p	0	0	0	902	1	0	0	0	0	0	0	0
streaming	0	0	1	1	4,405	18	0	0	0	0	0	0
voip	36	0	1	0	9	81,313	0	0	0	0	0	168
vpnChat	0	0	0	0	0	0	5,384	0	0	1	0	0
vpnEmail	0	0	0	0	0	0	0	468	0	0	0	0
vpnFileTransfer	0	1	0	0	0	0	0	0	3,178	0	1	0
vpnP2P	0	0	0	0	0	0	0	0	1	523	0	0
vpnStreaming	0	2	0	0	0	0	0	0	0	0	2,927	0
vpnVoip	8	0	1	0	0	202	0	0	1	0	0	16,997

4.4.2.2. Packet-Based Classification

Without any data sampling, because each step of packet-based classification takes a considerably long time, a single test is performed for Scenario A2 VPN only. Elapsed times are shown in Table 4.31.

Table 4.31. ML (Packet, no sampling) elapsed times

Scenario	**Data Generation**	**Training**		**Testing**	
		OneR	**J48**	**OneR**	**J48**
A2 VPN	3250.38	43.86	184.01	538.14	2118.99

All accuracy results for Scenario A2 VPN are either 100 percent or 1, which appears questionable, and it may be a sign of overfitting. J48 pruned tree output shows that feature flowiatMin was able to handle classification on its own. One can observe how discriminating this feature is looking at its graph. Features idleMin, flowiatMax, and idleMax draw almost identical graphs, thus give perfect accuracies as well. Another test runs to see what happens without these four features. Accuracy results are displayed in Table 4.32, elapsed times can be found in Table 4.33. Accuracy lowers slightly; no dramatic changes occur except that OneR starts labeling samples as vpnVoip, see Table 4.34. Table 4.35 shows J48 confusion matrix when four features are removed.

Table 4.32. ML (Packet, no sampling) Scenario A2 VPN, 4 features removed

Scenario	**Accuracy**		**Precision**		**Recall**		**F-Measure**	
	OneR	**J48**	**OneR**	**J48**	**OneR**	**J48**	**OneR**	**J48**
A2 VPN with four features removed	99.6078	99.9439	0.996	0.999	0.996	0.999	0.996	0.999

Table 4.33. ML (Packet, no sampling) Scenario A2 VPN, 4 features removed

Scenario	**Training**		**Testing**	
	OneR	**J48**	**OneR**	**J48**
A2 VPN with four features removed	19.03	219.03	468.97	3790

Table 4.34. ML (Packet, no sampling) A2 VPN OneR, 4 features removed

Predicted / Actual	**vpnChat**	**vpnEmail**	**vpnFileTransfer**	**vpnP2P**	**vpnStreaming**	**vpnVoip**
vpnChat	76,779	0	0	0	0	9,261
vpnEmail	0	20,927	0	0	0	631
vpnFileTransfer	0	0	372,136	0	0	4,090
vpnP2P	0	0	0	421,294	0	724
vpnStreaming	0	0	0	0	1,509,859	4,236
vpnVoip	0	0	0	0	0	2,410,093

Table 4.35. ML (Packet, no sampling) Scenario A2 VPN J48, 4 features removed

Predicted / Actual	**vpnChat**	**vpnEmail**	**vpnFileTransfer**	**vpnP2P**	**vpnStreaming**	**vpnVoip**
vpnChat	85,381	68	237	47	70	237
vpnEmail	78	21,408	28	10	7	27
vpnFileTransfer	202	42	375,421	63	217	281
vpnP2P	25	2	36	421,899	26	30
vpnStreaming	48	8	116	35	1,513,714	174
vpnVoip	177	25	240	24	132	2,409,495

Packet-based results after sampling are even better than flow-based results. OneR seems to perform excellently, with all accuracies being 100 percent. Feature flowiatMin is again very discriminative between the labels. It appears as if this single feature is enough to do both flow-based and packet-based classification perfectly. Table 4.36 consists of average accuracy results. Table 4.37 exhibits average elapsed times.

Table 4.36. ML (Packet) average accuracy results

Scenario	**Accuracy**		**Precision**		**Recall**		**F-Measure**	
	OneR	**J48**	**OneR**	**J48**	**OneR**	**J48**	**OneR**	**J48**
A1	100	99.99	0.9987	1	1	1	1	1
A2 non-VPN	100	99.996	1	1	1	1	1	1
A2 VPN	100	99.998	1	1	1	1	1	1
B	100	99.9965	1	1	1	1	1	1

Table 4.37. ML (Packet) average elapsed times

Scenario	**Data Generation**	**Training**		**Testing**	
		OneR	**J48**	**OneR**	**J48**
A1	53.33	0.17	0.17	1.43	1.03
A2 non-VPN	69.92	0.58	0.91	3.82	6.49
A2 VPN	26.02	0.68	0.94	3.92	8.06
B	101.12	1.21	2.55	11.59	27.45

Table 4.38 and 4.39 show average accuracy results and elapsed times when four features are removed.

Table 4.38. ML (Packet) average accuracy results, 4 features removed

Scenario	**Accuracy**		**Precision**		**Recall**		**F-Measure**	
	OneR	**J48**	**OneR**	**J48**	**OneR**	**J48**	**OneR**	**J48**
A1	99.733	99.9925	0.9974	1	0.9974	1	0.9974	1
A2 non-VPN	99.1325	-	0.9918	-	0.9913	-	0.9913	-
A2 VPN	99.0282	99.3968	0.9908	0.994	0.9904	0.9940	0.9904	0.994
B	98.6972	-	0.9888	-	0.987	-	0.9872	-

Table 4.39. ML (Packet) average elapsed times, 4 features removed

Scenario	**Data Generation**	**Training**		**Testing**	
		OneR	**J48**	**OneR**	**J48**
A1	53.33	0.09	0.12	0.91	0.88
A2 non-VPN	69.92	0.22	-	2.03	-
A2 VPN	26.02	0.24	0.73	2.76	8.27
B	101.12	0.65	-	12.35	-

With four features removed, average confusion matrix results for Scenario A1 OneR and J48 are shown in Table 4.40 and 4.41. For Scenario A2 non-VPN, they are shown in Table 4.42 and 4.43. Table 4.44 and 4.45 displays Scenario A2 VPN results. Finally, Table 4.46 and 4.47 refers to Scenario B.

Table 4.40. ML (Packet) A1 OneR, 4 features removed if accuracy is 1

Actual \ Predicted	**VPN**	**non-VPN**
VPN	9,998.2	1.8
non-VPN	51.6	9,948.4

Table 4.41. ML (Packet) A1 J48, 4 features removed if accuracy is 1

Actual \ Predicted	**VPN**	**non-VPN**
VPN	9,999.6	0.4
non-VPN	1	9,999

Table 4.42. ML (Packet) A2 non-VPN OneR, 4 features removed if accuracy is 1

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip
chat	9,946.6	0	0	53.4	0	0
email	35.4	9,851	0	113.6	0	0
fileTransfer	2.8	0	9,947.6	49.6	0	0
p2p	104.4	0	0	9,895.6	0	0
streaming	16.8	0	0	58.8	9,924.4	0
voip	26.4	0	0	32.8	0	9,940.8

Table 4.43. ML (Packet) A2 non-VPN J48, 4 features removed if accuracy is 1

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip
chat	9,999.6	0	0.4	0	0	0
email	0.8	9,999.2	0	0	0	0
fileTransfer	0	0.6	9,999.4	0	0	0
p2p	0	0	0	10,000	0	0
streaming	0.2	0	0	0	9,999.8	0
voip	0.2	0	0.2	0	0	9,999.6

Table 4.44. ML (Packet) A2 VPN OneR, 4 features removed if accuracy is 1

Actual \ Predicted	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
vpnChat	9,735.4	0	53.4	0	0	180.4
vpnEmail	127.8	9,787	42.6	0	0	42.6
vpnFileTransfer	10.2	0	9,985.8	0	0	4
vpnP2P	9.4	0	1.6	9,978.2	0	10.8
vpnStreaming	14	0	13	0	9,960.6	12.4
vpnVoip	57.8 0	2.8	0	0	0	9,939.4

Table 4.45. ML (Packet) A2 VPN J48, 4 features removed if accuracy is 1

Actual \ Predicted	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
vpnChat	9,946	2	19	0.6	0.8	0.8
vpnEmail	1	9,991.2	7.8	0	0	0
vpnFileTransfer	16.8	11.6	9,968.4	0.4	2.8	0
vpnP2P	0.4	0.6	0.6	9,998.4	0	0
vpnStreaming	1.2	0.2	4	0	9,994.4	0.2
vpnVoip	1.2	0.4	0.8	0	0.2	9,997.4

Table 4.46. ML (Packet) B OneR, 4 features removed if accuracy is 1

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
chat	9,719.8	0	0	0	0	0	156	0	0	0	0	124.2
email	0	9,851	0	0	0	0	132	0	0	0	0	17
fileTransfer	0	0	9,947.6	0	0	0	50.2	0	0	0	0	2.2
p2p	0	0	0	9,739	0	0	208.8	0	0	0	0	52.2
streaming	0	0	0	0	9,924.4	0	61.8	0	0	0	0	13.8
voip	0	0	0	0	0	9,940.8	36.2	0	0	0	0	23
vpnChat	0	0	0	0	0	0	9,819.6	0	0	0	0	180.4
vpnEmail	0	0	0	0	0	0	170.4	9,787	0	0	0	42.6
vpnFileTransfer	0	0	0	0	0	0	166.2	0	9,829.8	0	0	4
vpnP2P	0	0	0	0	0	0	11	0	0	9,978.2	0	10.8
vpnStreaming	0	0	0	0	0	0	27	0	0	0	9,960.6	12.4
vpnVoip	0	0	0	0	0	0	61.2	0	0	0	0	9,938.8

Table 4.47. ML (Packet) B J48, 4 features removed if accuracy is 1

Actual \ Predicted	chat	email	fileTransfer	p2p	streaming	voip	vpnChat	vpnEmail	vpnFileTransfer	vpnP2P	vpnStreaming	vpnVoip
chat	9999	0	0.4	0	0	0	0.6	0	0	0	0	0
email	0.8	9998.8	0	0	0	0	0.4	0	0	0	0	0
fileTransfer	0	0	9999.4	0	0	0	0	0	0.4	0	0	0.2
p2p	0	0	0	10000	0	0	0	0	0	0	0	0
streaming	0.4	0	0	0	9999.6	0	0	0	0	0	0	0
voip	0	0	0	0	0	9999.6	0	0	0	0	0	0.4
vpnChat	0	0	0	0	0	0	10000	0	0	0	0	0
vpnEmail	0	0	0	0	0	0	0	10000	0	0	0	0
vpnFileTransfer	0	0	0	0	0	0	0	0	9999.8	0	0.2	0
vpnP2P	0	0	0	0	0	0	0	0	0	10000	0	0
vpnStreaming	0	0.4	0	0	0	0	0	0	0	0	9999.6	0
vpnVoip	0	0	0	0	0	0	0	0	0	0	0	10000

4.5. EVALUATION

In the following charts (Fig. 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17); LSTM-FS (Offline), LSTM-FS (Online) precision and recall values are shown together with the work (C4.5) by Draper-Gil et al. [1]. Flow-based and packet-based ML-SF methods are not included in the comparison graphs since even though ML-SF is designed based on C4.5, the results were almost 100 percent for each scenario. Both studies use a flow timeout value of 15 seconds. The difference may be because of the different pre-labeling of the samples; they have an additional label named browsing. They did only flow-based classification. They got the best accuracy with the C4.5 algorithm. The purple columns in the charts correspond to their work; the values are from their paper, but there may be slight differences as they did not share the exact values. The blue columns represent offline LSTM results, while the green ones are online LSTM results. The reason for choosing precision and recall metrics is that Draper-Gil et al. only presented these values. They displayed the values per label: BRW (browsing), CHAT, STR (streaming), MAIL, VOIP, P2P, and FT (file transfer). Label BRW is left out from the graphs since our work does not contain this label. It can be generalized that offline LSTM performs the best, followed by online LSTM. Please note that the comparison between our method and Draper-Gil et al.'s could be fairer if we used the same labeling. Even though we used the same dataset, this was not possible since they did not share how they labeled each PCAP file. For example, it is unclear which PCAPs they associated with the label BRW.

For Scenario A1, it is clear that this study gives much better results than theirs, and there is not much difference between offline/online LSTM. As precision is related to false positives, we can infer that all three algorithms are prone to mislabeling as VPN where the actual label is non-VPN. On the other hand, a higher recall means a lower number of false negatives, which suggests that all three algorithms are better at identifying VPN traffic than non-VPN. Since there are only two labels for this scenario, precision and recall gave similar meaning results.

Scenario A-2 precision results show that except for VOIP and P2P our both algorithms excelled (especially for CHAT, STR, and MAIL). Offline LSTM is generally better than online; for FT, online LSTM has the best result. The difference between both algorithms stands out for VOIP and P2P. Furthermore, for P2P, C4.5 performs better than online LSTM. Label CHAT seems to have more false positives than the other labels for all algorithms. Regarding recall results, online LSTM now beat C4.5 for P2P. It means that even though C4.5 gives fewer false positives, it gives more false negatives for P2P. Interestingly, for VOIP, C4.5's recall result is better than both of our algorithms'. C4.5 is especially good at differentiating VOIP among other non-VPN labels.

Offline LSTM stands out for its Scenario A2 VPN-CHAT precision result, whereas online LSTM performed the worst. For the rest of the labels, our algorithms performed much better than C4.5. However, VPN-VOIP results are pretty close. VPN-MAIL recall result for online LSTM is especially low.

C4.5 VOIP and VPN-VOIP precision performance are surprisingly not good for Scenario B, even though it excelled for Scenario A2 non-VPN and VPN. It can be said that when non-VPN and VPN labels are mixed C4.5 struggles to identify correctly. Both algorithms seem to be better at this. Online LSTM especially struggles with P2P and VPN-FT. C4.5 VOIP and VPN-VOIP recall results are still good; however, none of the recall results beat our algorithms.

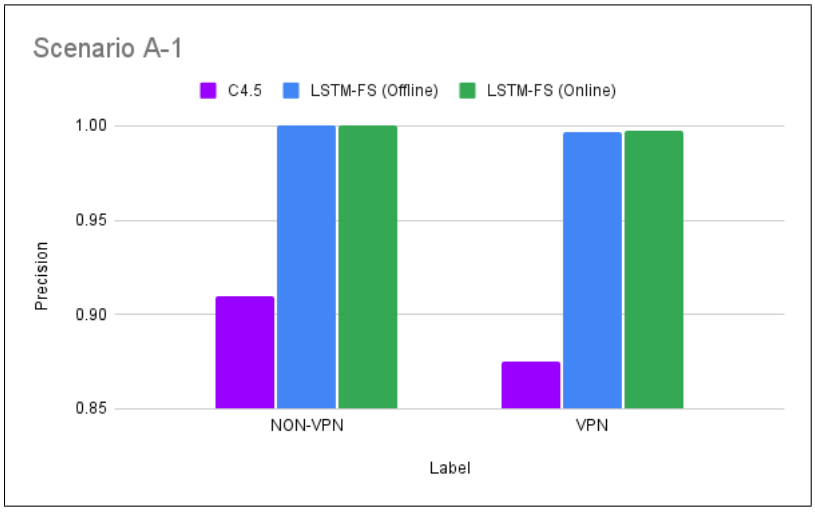

Figure 4.10. Precision comparison of [1] and LSTM for Scenario A1

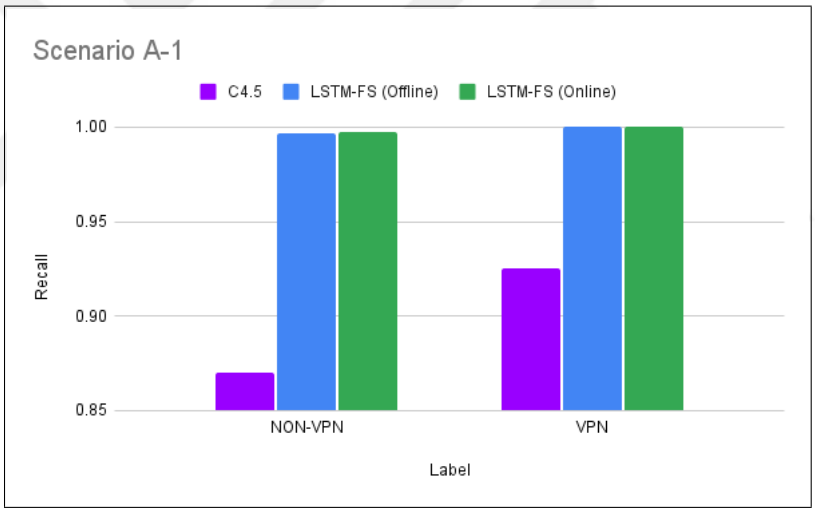

Figure 4.11. Recall comparison of [1] and LSTM for Scenario A1

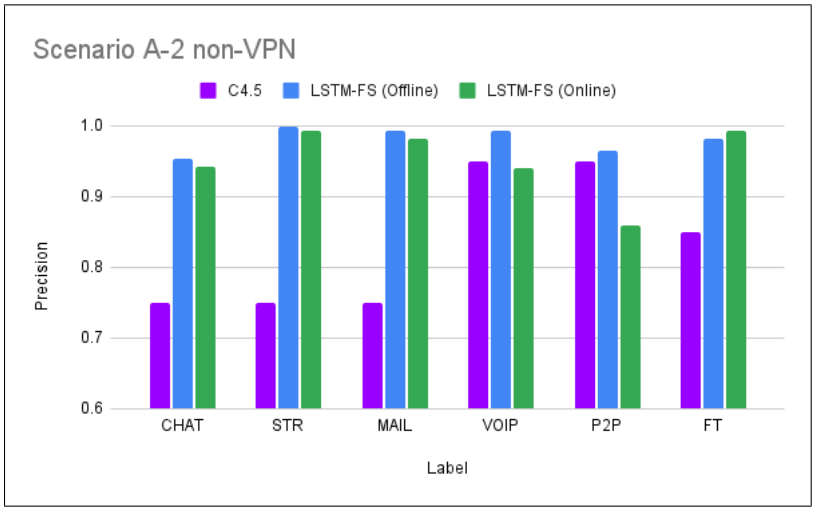

Figure 4.12. Precision comparison of [1] and LSTM for Scenario A2 non-VPN

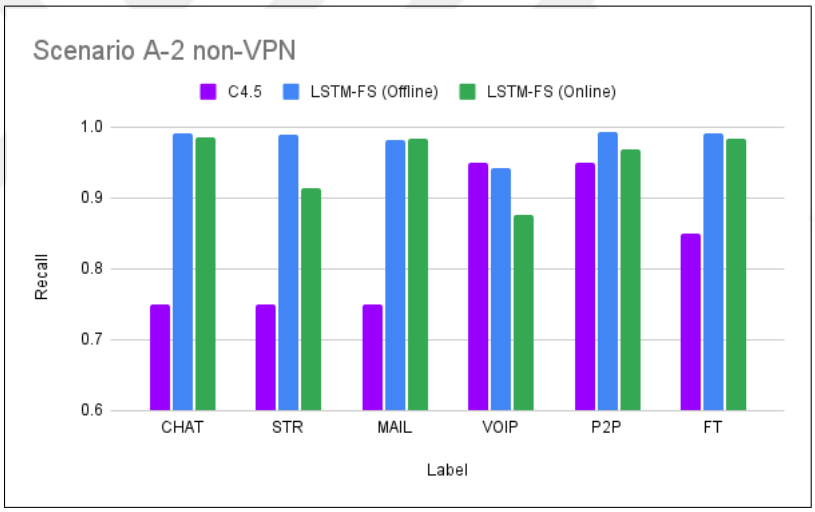

Figure 4.13. Recall comparison of [1] and LSTM for Scenario A2 non-VPN

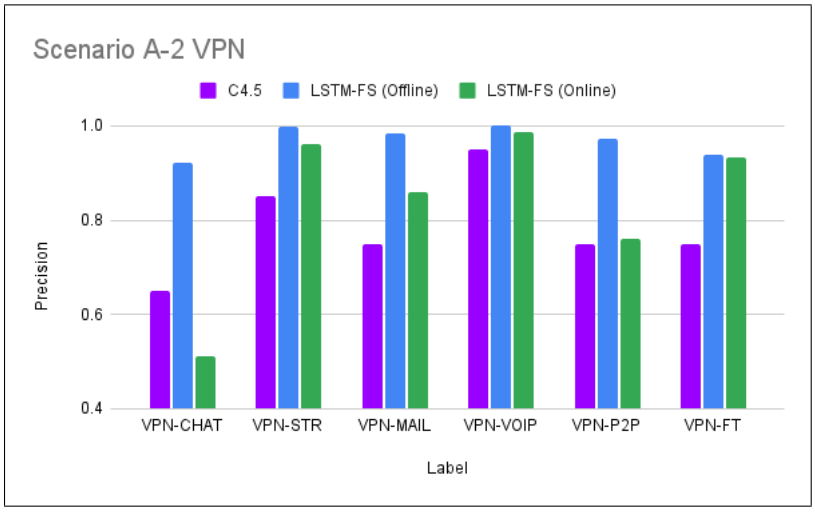

Figure 4.14. Precision comparison of [1] and LSTM for Scenario A2 VPN

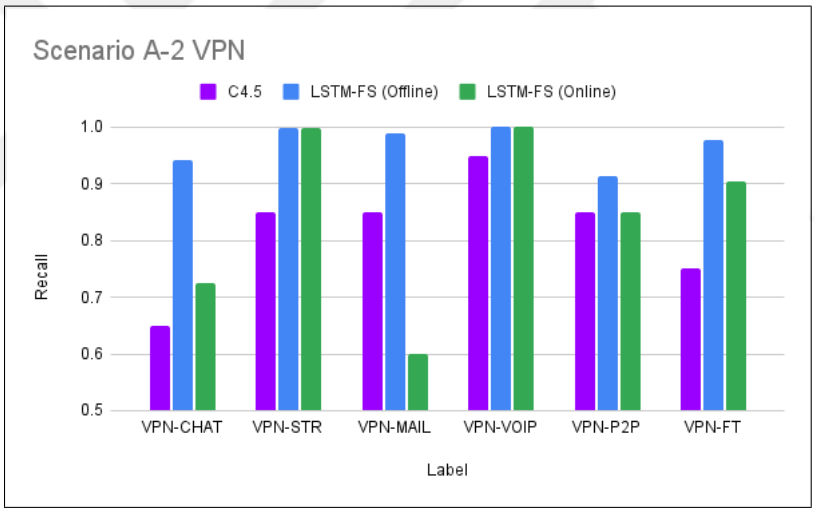

Figure 4.15. Recall comparison of [1] and LSTM for Scenario A2 VPN

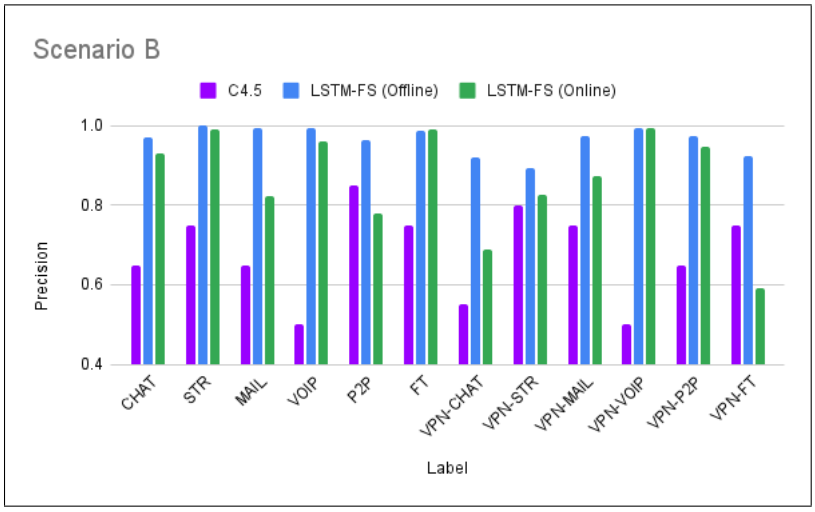

Figure 4.16. Precision comparison of [1] and LSTM for Scenario B

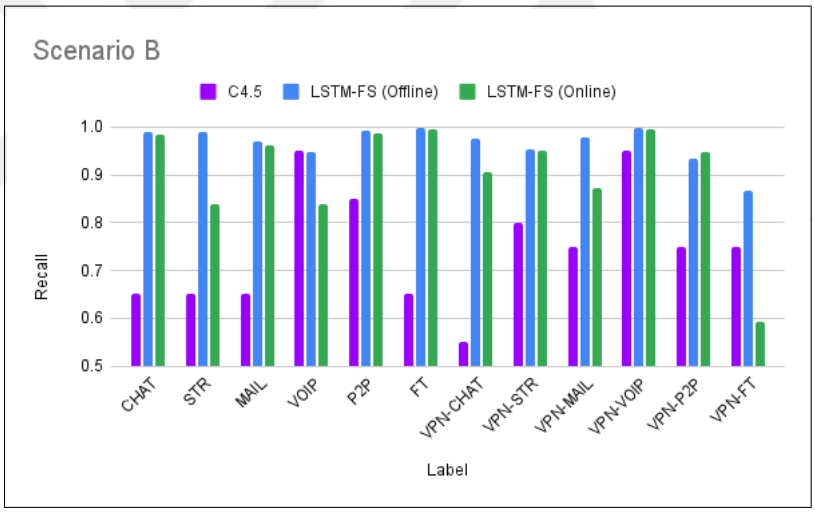

Figure 4.17. Recall comparison of [1] and LSTM for Scenario B

One question that may arise regarding test results is why does Scenario A2 VPN perform the worse. A critical success factor for a VPN application is how well it disguises its users' behavior. That is, VPN providers make an effort to hide the real traffic happening. One way to achieve this is to mimic irrelevant traffic. For instance, users' traffic can be altered to look like commonly-used websites' traffic. In that case, it is understandable why the learning model mispredicts labels.

Byte Segment Neural Network [32] spends 43 ms for each data sample at the training phase, and the total training time is 123 minutes. On average, training takes at most 68.29 seconds with LSTM-FS for Scenario B, 2.15 seconds with ML-SF One-R for Scenario A1, and 13.2 seconds with ML-SF C4.5 for Scenario A1. Thus processing overhead per sample is 0.5 ms, 0.1 ms, and 0.7 ms, respectively, since Scenario B consists of 120,000 data and Scenario A1 has 20,000. On the other hand, they measured the average time to deal with a sample during online classification as 2.97 ms and shared that for Securitas [33] this is 7.01 ms. Packet processing overhead is at most 0.7477 ms with LSTM-FS for Scenario A2 non-VPN. Packet-based ML-SF One-R and C4.5 take 11.59 and 2745 ms to test 120,000 samples for Scenario B. Thus there exists 0.1 ms and 0.2 ms overhead.

5. CONCLUSIONS

There are two methods presented in this study for packet categorization. One is an LSTM deep learning model trained with features that can be taken directly from a network packet's headers. There is no payload-level feature as in previous work. Another difference is that no packet gets discarded if its flow has more than the preset sequence length. The other method is a machine learning model with statistical features. The methods were tested with 14 classification labels: non-VPN, VPN, chat, email, file transfer, P2P, streaming, VoIP, VPN chat, VPN email, VPN file transfer, VPN P2P, VPN streaming, and VPN VoIP. While both methods yield valid results, outperforming a well-known machine learning study, the machine learning method came to the fore with both its speed and accuracy. It seems as if only a single feature (any of flowiatMin, flowiatMax, idleMin, and idleMax) is enough to classify labels of the four scenarios. This method was made by adapting [1], despite this, gave much better results than it. Compared to the only paper that shared its online classification results, the LSTM model presented in this paper gave better results in terms of speed.

The programs presented in this study can be easily used not only for categorization but also for other types of traffic classification (e.g., protocol or application classification). All that needs to be done is to pre-label the input data correctly and introduce these labels to the programs. In addition, a new feature can be easily added to both programs, or an existing feature can be removed. For these reasons, it can be said that this study will be instrumental in paving the way for many future studies.

REFERENCES

1. Habibi Lashkari A, Draper Gil G, Mamun M, Ghorbani A. Characterization of Encrypted and VPN Traffic Using Time-Related Features. *In: The International Conference on Information Systems Security and Privacy (ICISSP)*; 2016. .

2. Traffic classification;. [cited 2021 25 Sep]. https://en.wikipedia.org/wiki/Traffic_classification.

3. Internet Traffic Classification;. [cited 2021 25 Sep]. https://www.sandvine.com/hubfs/downloads/archive/technology-showcase-internet-traffic-classification.pdf.

4. Tackling online safety with DPI-enhanced parental controls;. [cited 2021 25 Sep]. https://www.ipoque.com/blog/online-safety-dpi-enhanced-parental-control.

5. Advanced Traffic Classification;. [cited 2021 25 Sep]. https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2020/Whitepapers/Sandvine_WP_Advanced%20Traffic%20Classification.pdf.

6. Government Cyber Defense;. [cited 2021 25 Sep]. https://www.qosmos.com/cybersecurity/dpi-sensor-for-cyber-defense/.

7. R&S PACE 2;. [cited 2021 25 Sep]. https://www.ipoque.com/products/dpi-engine-rs-pace-2.

8. Advanced Deep Packet Inspection;. [cited 2021 25 Sep]. https://www.ipoque.com/solutions/deep-packet-inspection-for-software-vendors.

9. Service Name and Transport Protocol Port Number Registry;. [cited 2021 26 June]. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

10. Chapter: Traffic Classification;. [cited 2021 26 June]. https://www.cisco.com/c/en/us/td/docs/nsite/enterprise/wan/wan_optimization/wan_opt_sg/chap05.html.

11. HTTPS encryption on the web;. [cited 2021 26 June]. https://transparencyreport.google.com/https/overview?hl=en.

12. The Relevance of Network Security in an Encrypted World;. [cited 2021 26 June]. https://blogs.vmware.com/networkvirtualization/2020/09/network-security-encrypted.html/.

13. First Packet Classification in an Encrypted World;. [cited 2021 25 Sep]. https://www.thefastmode.com/expert-opinion/20216-first-packet-classification-in-an-encrypted-world.

14. Rezaei S, Liu X. Deep Learning for Encrypted Traffic Classification: An Overview. *IEEE Communications Magazine*. 2019 May;57(5):76–81. Available from: http://dx.doi.org/10.1109/MCOM.2019.1800819.

15. Wireshark;. [cited 2021 20 July]. https://www.wireshark.org/.

16. VPN-nonVPN dataset (ISCXVPN2016);. [cited 2021 26 June]. https://www.unb.ca/cic/datasets/vpn.html.

17. Yamansavascilar B, Guvensan A, Yavuz A, Karsligil E. Application identification via network traffic classification. *In: International Conference on Computing, Networking and Communications (ICNC)*; 2017. p. 843–848.

18. SPID Statistical Protocol IDentification;. [cited 2021 27 June]. https://sourceforge.net/projects/spid/.

19. Hjelmvik E. The SPID Algorithm Statistical Protocol IDentification; 2008. .

20. Hjelmvik E, John W. Statistical Protocol IDentification with SPID: Preliminary Results. 2009 01.

21. Bagui S, Fang X, Kalaimannan E, Bagui SC, Sheehan J. Comparison of machine-learning algorithms for classification of VPN network traffic flow using time-related features. *Journal of Cyber Security Technology*. 2017;1(2):108–126.

22. Wang W, Zhu M, Wang J, Zeng X, Yang Z. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. *In: 2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*; 2017. p. 43–48.

23. Deep Learning models for network traffic classification;. [cited 2021 13 June]. https://github.com/echowei/DeepTraffic.

24. USTC-TK2016;. [cited 2021 13 July]. https://github.com/yungshenglu/USTC-TK2016.

25. Lotfollahi M, Zade RSH, Siavoshani MJ, Saberian M. Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning; 2018.

26. Parchekani A, Naghadeh SN, Shah-Mansouri V. Classification of Traffic Using Neural Networks by Rejecting: a Novel Approach in Classifying VPN Traffic; 2020.

27. Zhou K, Wang W, wu C, Hu T. Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks. *ETRI Journal*. 2020 01;42.

28. Tor-nonTor dataset (ISCXTor2016);. [cited 2021 17 July]. https://www.unb.ca/cic/datasets/tor.html.

29. Lopez-Martin M, Carro B, Sanchez-Esguevillas A, Lloret J. Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access*. 2017 09;PP.

30. RedIRIS - Welcome to RedIRIS;. [cited 2021 18 July]. `https://www.rediris.es/`.

31. Deri L, Martinelli M, Bujlow T, Cardigliano A. nDPI: Open-source high-speed deep packet inspection. *In: 2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*; 2014. p. 617–622.

32. Li R, Xiao X, Ni S, Zheng H, Xia S. Byte Segment Neural Network for Network Traffic Classification. *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 2018:1–10.

33. Yun X, Wang Y, Zhang Y, Zhou Y. A Semantics-Aware Approach to the Automated Network Protocol Identification. *IEEE/ACM Transactions on Networking*. 2016;24(1):583–595.

34. PcapAnalyzer;. [cited 2021 20 July]. `https://github.com/denizergonul/PcapAnalyzer`.

35. ISCXFlowMeter;. [cited 2021 20 July]. `https://github.com/ahlashkari/ISCXFlowMeter`.

36. TCPDUMP/LIBPCAP public repository;. [cited 2021 20 July]. `https://www.tcpdump.org/`.

37. tcprewrite;. [cited 2021 29 August]. `https://tcpreplay.appneta.com/wiki/tcprewrite`.

38. Unified Modeling Language, v2.5.1;. [cited 2021 20 July]. `https://www.omg.org/spec/UML/2.5.1/PDF`.

39. et al A. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems; 2015. Software available from tensorflow.org. Available from: https://www.tensorflow.org/.

40. Chollet F, et al.. Keras; 2015. https://keras.io.

41. CppFlow;. [cited 2021 23 July]. https://github.com/serizba/cppflow.

42. Witten IH, Frank E, Hall MA, Pal CJ. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*. 4th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2016.