Title: Real-Time Encrypted Traffic Classification with Deep Learning

Authors: Deniz Tuana ERGÖNÜL, Onur DEMİR

Recieved: 2021-11-23 00:00:00

Accepted: 2022-03-08 00:00:00

Article Type: Research Article

Volume: 26
Issue: 2
Month: April
Year: 2022
Pages: 313-332

How to cite
Deniz Tuana ERGÖNÜL, Onur DEMİR; (2022), Real-Time Encrypted Traffic Classification with Deep Learning. Sakarya University Journal of Science, 26(2), 313-332, DOI: 10.16984/saufenbilder.1026502
Access link
https://dergipark.org.tr/tr/journal/1115/issue/69580/1026502

FEN BİLİMLERİ ENSTİTÜSÜ DERGİSİ
Sakarya University Journal of Science (SAUJS)
Cilt/Volume: 26
Sayı/Issue: 2
Nisan/April 2022

Sakarya University Journal of Science 26(2), 313-332, 2022

Real-Time Encrypted Traffic Classification with Deep Learning

Deniz Tuana ERGÖNÜL*[1], Onur DEMİR[1]

Abstract

Confidentiality requirements of individuals and companies led to the dominance of encrypted payloads in the overall Internet traffic. Hence, traffic classification on a network became increasingly difficult as it must rely on only the packet headers. Many vital tasks such as differential pricing, providing a safe Internet for children, and eliminating malicious connections require traffic classification, even if the payload contents are encrypted. Encrypted traffic is harder to classify as packet content becomes unreadable. In this work, we aim to provide an insight into traffic classification using encrypted packets in terms of both accuracy and packet processing time. LSTM (Long Short-Term Memory) architecture is a good candidate for this problem as it can handle sequences. Each flow can be modeled as a sequence and patterns of the sequences can provide valuable information. We compare the performance of LSTM with other methods in both real-time and offline experiments. Compared to a machine learning method both online and offline LSTM excelled with precision and recall differences up to 50%. Average accuracy with LSTM was measured as 97.77% offline and 91.7% in real-time. Average packet processing time in real-time was recorded as 0.593 msec which is 5 times faster than a recent work that uses LSTM method.

Keywords: Deep learning, neural networks (Computer), computer communication networks, classification

1. INTRODUCTION

In traffic classification, the principal aim is to recognize and name different groups of packets using the information available at the network level. 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, detecting malicious user activities, or dropping certain content for parental control [1]. Classification can be in terms of the application, protocol, or category of a packet [2]; for example, WhatsApp Web application, HTTPS protocol, chat category; Skype application, TLS protocol, VoIP category. The need for classification comes from different reasons such as 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

* Corresponding author: dtergonul@gmail.com
[1] Yeditepe University, Faculty of Engineering
E-mail: odemir@cse.yeditepe.edu.tr
ORCID: https://orcid.org/0000-0003-2945-0833, https://orcid.org/0000-0002-1088-6461

defense industry units of countries; however, every institution can benefit from a dedicated solution whether it is online or offline [3]. Labeling of pre-captured network packets is called offline classification; online classification is when they are labeled one after another in real-time [4].

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 [5]. Although practical in respect of time and effort, this technique is not very reliable. The applications can pick other ports to communicate with each other rather than using the standard ports. Usage of nonstandard ports jeopardizes the reliability of this method. There is even a term called port obfuscation [6]. 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 [7] (Figure 1) shows that the preference for HTTPS over HTTP has increased significantly in the last seven years, from 48% in 2014 to 95% 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.

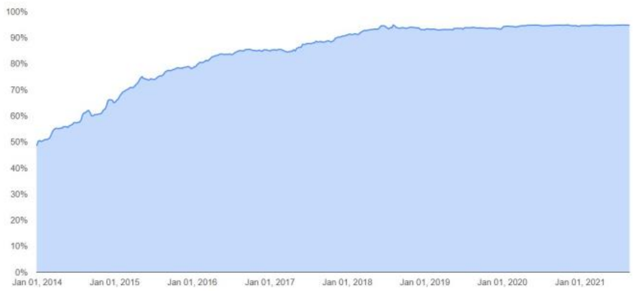

Figure 1 Encrypted traffic across Google

Statistical flow classification is another valid method that uses features such as average packet inter-arrival times, packets per second, byte frequencies, etc. [6].

In recent years, research on machine learning and deep learning models intensified, since the accumulation and storage of data has become easier and cheaper. The advantages of these models are that they can be trained offline with an abundant amount of training data. As it is for many learning problems, feature selection may be considered a tedious task, but it is possible to make a model learn how to do that on its own. Deep learning is a relatively new technology for flow classification. There are not many research papers yet, especially with an LSTM model, and the existing ones usually do offline classification [8]. Thus, the elapsed time, 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 [9] can collect live traffic into packet capture (PCAP) files. The difficulty here is that the collected data is often noisy. The packets which do not fall in any of the main categories are noise. 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) [10] 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 [11].

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, 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.

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

- Why is LSTM a good candidate for the traffic classification problem?
- How to model this problem for LSTM?
- How does LSTM perform both accuracy and timewise?

The reason for choosing LSTM is that it has a memory where earlier sequences of packets can be remembered. 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 has been 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. 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.

There are four scenarios to test LSTM-FS. The first scenario is to detect whether the traffic is VPN or not. In the second scenario, six non-VPN categories (chat, email, file transfer, P2P, streaming, VoIP) exist to label the given data. The third scenario only includes VPN versions of these six categories. In the last scenario, with the second and third scenarios combined, labeling is made among six non-VPN and six 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.

Section 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 problem is explained along with the presented solution in Section 3. Section 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 Section 5; possible improvements and new ideas for future work are also discussed.

2. BACKGROUND

In this section, we tried to compile a comprehensive list of related studies to our work. One of the studies included in this section is a purely statistical method, while three of them consist of machine learning models trained with statistical features. There is a survey paper about deep learning methods that also underlines the difficulties of encrypted traffic classification. The remaining works are all deep learning methods, and two of which are LSTM. Most of the classification involves grouping based on protocols, applications, or other categories. Only the last study mentions real-time classification. ISCXVPN2016 [10] stands out as the major dataset used.

Statistical Protocol Identification (SPID) [12-14] 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 stores the 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 one 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. Each meter calculation returns a list of indices for the meter’s counter vector. The elements at these indices are increased by 1. The researchers got an average of 100% precision and 92% recall when tested with BitTorrent, eDonkey, HTTP, SSL, and SSH flows.

Characterization of Encrypted and VPN Traffic Using Time-Related Features [15] 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%.

Yamansavascilar et al. [11] 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 we do 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, but the selection procedure of these features is not unclear. Accuracy varies between 85.44%-93.94% for the external dataset and 69.90%-90.87% 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. [16] compared the performance of six supervised machine learning techniques: Logistic Regression, Support Vector Machine (SVM), Naive Bayes, kNN, Gradient Boosting Trees, and Random Forest, in terms of accuracy, precision, sensitivity, and specificity. They utilized the same 24 time-related features in [15] 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%-50%). 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%. 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 deteriorates accuracy results to 90%, they claim that it can be advantageous where speed matters more, but they did not present any experiment results backing this assumption.

One-dimensional Convolutional Neural Network (1-D CNN) is a model for traffic identification and classification. Wang et al. [17] claimed network traffic as sequential data. They argue that solution becomes more prone to get stuck at a local minimum when a problem is divided into sub-problems. Consequently, they present an end-to-end solution where feature extraction, feature selection, and classifier are one [16]. 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 [19] 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 addressed 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%. They also pointed out that 1-D CNN delivers better performance than two-dimensional.

Rezaei and his team [8] 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. They state that other than ISCXVPN2016 and a few others, there is not a publicly available and widely used dataset. 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 circumstances (i.e., high congestion). They also stressed how to tackle encrypted traffic. For example, time-series features such as inter-arrival time are not changed 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 such as 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 [20] can identify the protocol or application to which a packet belongs (by 98% recall), label it by its category while differentiating VPN from non-VPN (94%). 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. [21] 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%), 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 accomplish this, they use other thresholds to decide if another classification is necessary.

With his team, Zhou [22] 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 works, results improved by 1% to 7%, with Random Forest remaining the best method (98%). Additionally, they built a NN which takes 23 statistical features (from [15]) as input to classify Tor data (ISCXTor2016 [23]) into 8 categories, as opposed to 7 seen in prior works. (Streaming is divided into two labels: audio and video) With this model, they improved some of the previous studies' metrics by nearly 30%. 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. [24] integrated LSTM on top of CNN to classify a dataset from RedIRIS [25], which contains 266,160 flows with 108 applications. They used nDPI [26], 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% and SSL with 15%. After trying out a few combinations, they settled on five features: source port, destination port, payload length, TCP window size, and packet direction. (96% 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 employed 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%.

Byte Segment Neural Network is a model by Liu et al. [27] 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 length 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 [28], they got better results for 5 of the applications considering recall, precision, and F-score. They have an average F-score of 95.82%. In the training phase, it takes 43 msec on average to prepare input, and 20 epochs of training take 123 minutes, whereas, in the testing phase, processing a datagram takes 2.97 msec.

Table 1 Related Work

Paper	Classification	Method	Features
Statistical Protocol IDentification with SPID: Preliminary Results [14]	Protocol	Statistical	7 flow-level, 27 payload-level
Characterization of Encrypted and VPN Traffic Using Time-Related Features [15]	Category	k-NN, C4.5 (J48)	24 time-based
Application Identification via Network Traffic Classification [11]	Application	J48, Random Forest, k-NN, Bayes Net	111 (Reduced to 12)
Comparison of machine-learning algorithms for classification of VPN network traffic flow using time-related features [16]	Category	Logistic Regression, SVM, Naive Bayes, k-NN, Gradient Boosting Trees, Random Forest	24 time-based
End-to-end Encrypted Traffic Classification with One-dimensional Convolution Neural Networks [17]	Category	1-D CNN	First 784 bytes of a flow
Deep Packet: A Novel Approach for Encrypted Traffic Classification Using Deep Learning [20]	Protocol, Application, Category	Stacked autoencoders + 1-D CNN	First 1500 bytes of a packet
Classification of Traffic Using Neural Networks by Rejecting: A Novel Approach in Classifying VPN Traffic [21]	Category	MLP + RNN	First 784 bytes of a flow
Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks [22]	Category	Entropy estimation + (SVM, Random Forest, Naive Bayes, Logistic Regression, NN)	23 time-based
Network Traffic Classifier with Convolutional and Recurrent Neural Networks for Internet of Things [24]	Protocol, Application	CNN, RNN, CNN + LSTM	5
Byte Segment Neural Network for Network Traffic Classification [27]	Protocol, Application	LSTM, GRU	Equal length byte segments

Table 1 consists of all the studies summarized except Deep Learning for Encrypted Traffic Classification [8], 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.

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 msec as the average datagram processing time; our proposed solution LSTM-FS performs 5 times faster than their solution.

3. METHODOLOGY

In this section, our proposed solution for traffic classification, LSTM with Flow Sequences (LSTM-FS), is described. The section is divided into five subsections for flow generation (3.1.), feature extraction (3.2.), creating train and test data (3.3.), model generation (3.4.), and testing offline/online classification (3.5.).

Our solution operates with PCAP files; and includes libpcap [29], a C/C++ library for network traffic capture operations. Wireshark [9], which is a well-known opensource network packet analyzer, also utilizes this library. The solution supports 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) protocols. 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 [30] so that their packets have fake Ethernet layers.

3.1. Flow Generation

The first step of our solution for the traffic classification problem is bidirectional flow generation; in other words, identifying the packets with the same endpoints and ports so that they are associated with the same flow. Since its five-tuple uniquely describes a flow, after a given packet is analyzed, it is grouped with the others which have the same L4 protocol, source-destination port, and IP pairs. In our design, a flow is strictly related to at least one non-dummy packet (A dummy packet's all features appear as -1.), and a classification label.

3.2. Feature Extraction

In our solution, packets hold almost everything one can obtain from a network packet except the payload. We exclude 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).

Packet fields shown in Table 2 directly correspond to features to feed into the learning model. Valid packet directions are forward and backward. TCP fields are all set to 0 for UDP packets and vice versa. It is 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.

Table 2 Packet fields

Field	**Definition**
direction	Packet direction
actualLength	Length on wire, frame length
arrivalTime	Epoch time
ipHeaderLength	IP header length (IHL)
ipTypeOfService	Type of Service (TOS) field of IP header
ipTotalLength	IP header length + IP data length (datagram length)
ipTimeToLive	Time to Live (TTL) field of IP header
tcpSrcPort	TCP source port
tcpDestinationPort	TCP destination port
tcpSeqNumber	Sequence number field of TCP header
tcpAckNumber	Acknowledgement number field of TCP header
tcpHeaderLength	TCP header length (HLEN, data offset)
tcpReserved	Reserved field of TCP header
tcpFlags	Flags field of TCP header
tcpWindowSize	Sliding window size field of TCP header
tcpUrgentPointer	Urgent pointer field of TCP header
udpSrcPort	UDP source port
udpDestinationPort	UDP destination port
udpLength	UDP header length (8 bytes) + UDP data length

3.3. Creating Train and Test Data

To create training and testing data, we first examine PCAP files under subdirectories of a given directory. Subdirectory names get interpreted as labels. After all the PCAPs get examined, sample creation will take place, as shown in Figure 2.

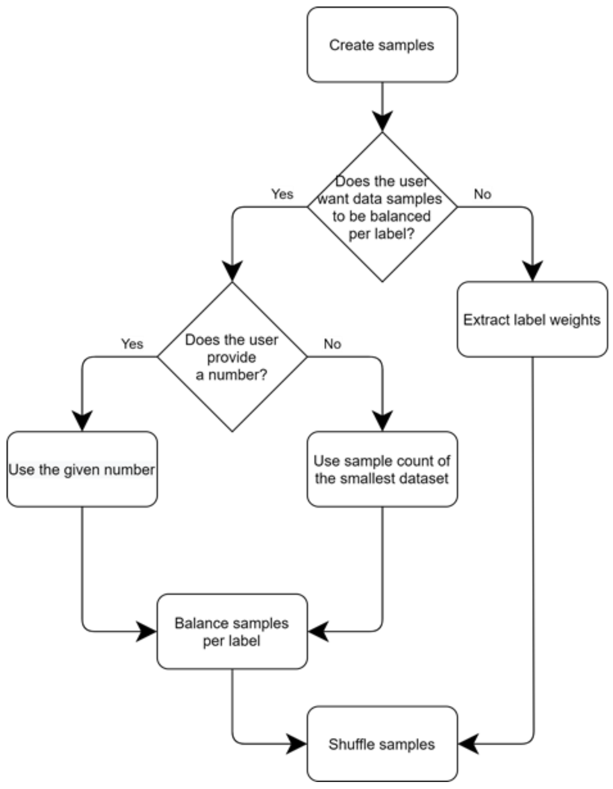
Create samples
Does the user want data samples to be balanced per label?
Yes
No
Does the user provide a number?
Yes
No
Extract label weights
Use the given number
Use sample count of the smallest dataset
Balance samples per label
Shuffle samples

Figure 2 Sample creation flowchart

For each flow, a set of samples get created. A sample is a flow sequence which is part of a flow and consists of at least one non-dummy packet. Sequences have a preset length of 10. Thus, flows with fewer packets get padded with dummy packets (Figure 3). On the other hand, for flows with more packets, extra samples are created. To be exact, every 10 consecutive packets in a flow becomes a sample (Figure 4). Users can choose whether they want the data to be balanced per label or not. This way, any deprivation an imbalanced dataset may cause (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. After samples are created, (optionally) balanced, then shuffled, 85% 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.

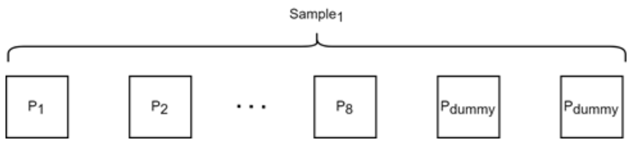

Figure 3 Sample creation when the number of packets in flow is less than 10

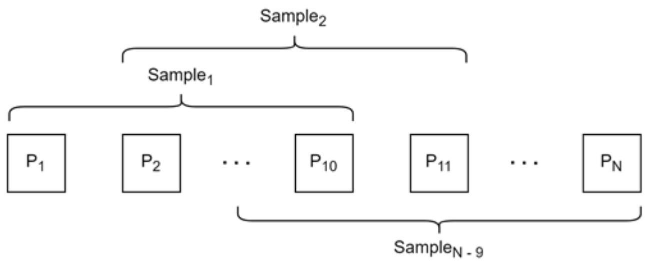

Figure 4 Sample creation when the number of packets in flow is greater than 10

Samples go through one more step before being written to the relevant file; feature scaling. Equation 1 shows min-max normalization (rescaling) formula where x_i is the original value for i^{th} sample, and x_i^0 is the normalized value. This requires finding the minimum and maximum values for each feature. Note we only consider TCP packets for the calculation for TCP-related features. Naturally, the same applies to UDP as well. Dummy packets are not considered for scaling. 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. Invalid features will not appear in training and testing files.

$$x_i' = \frac{x_i - \min(x)}{\max(x) - \min(x)} \qquad (1)$$

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 realistic simulation is required. Our outputs are training and test datasets as CSV files; the test data is also extracted as a single PCAP file (dummy packets excluded) where its packets are shuffled using Algorithm 1. This mixed data is more representative of an online setting than data in which flow sequences come in sizes of 10 one after another. Function nextPacketID ensures that packets stay in order in their flows (Figure 5). The new packet series gets dumped to a PCAP file beside a text file for each packet label.

```
 1: function shufflePackets
 2:     $packets \leftarrow []$
 3:     $availableFlowCount \leftarrow size(flows)$
 4:     while $availableFlowCount > 0$ do
 5:         $f \leftarrow random(availableFlowCount)$
 6:         $p \leftarrow nextPacketID(f)$
 7:         if $ID(p) \neq -1$ then
 8:             Add $p$ to $packets$
 9:         else                              ► End of flow
10:             $availableFlowCount \leftarrow availableFlowCount - 1$
11:             Remove $f$ from $flows$
12:         end if
13:     end while
14: end function
```

Algorithm 1 Algorithm to shuffle packets retaining order in their flows

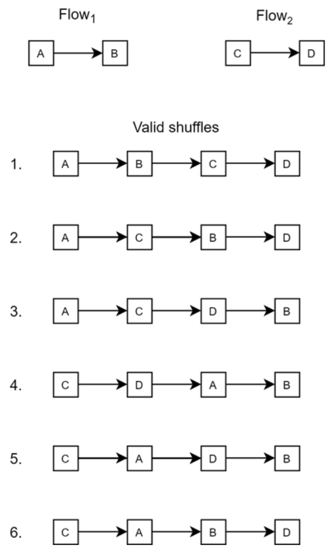

Figure 5 Valid shuffle examples

3.4. Model Generation

Our solution includes a Python program that uses TensorFlow [31] and Keras [32] libraries. The program builds a learning model, trains it with given data, and serializes it. The serialized model is imported through an API named CppFlow [33]. 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 its success with modeling sequential data, series of consecutive packets are also expected to fit in this model. 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.

Table 3 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, batch training, which reduces training time with the help of parallelism, must still contain equal length samples. For this reason, padding can be used. 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.

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. The required coefficients are outputted during train and test data creation. 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.

3.5. Testing Offline/Online Classification

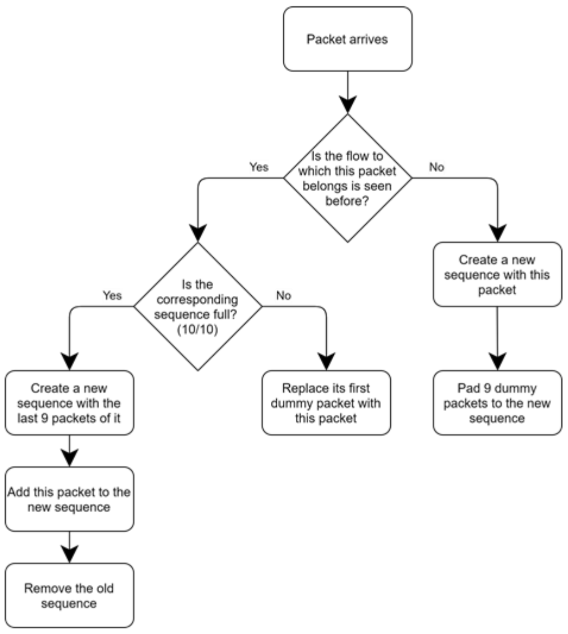
Packet arrives
Is the flow to which this packet belongs is seen before?
Yes
No
Is the corresponding sequence full? (10/10)
Yes
No
Create a new sequence with this packet
Create a new sequence with the last 9 packets of it
Replace its first dummy packet with this packet
Pad 9 dummy packets to the new sequence
Add this packet to the new sequence
Remove the old sequence

Figure 6 Sequence (sample) creation when testing

For this case, our solution accepts 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 normalization. We utilize actual packet labels for accuracy calculations, comparing expectations with predictions, and creating a confusion matrix. Since the learning model learns with a subset of features, the same subset must be obtained during testing. Similarly, the same scaling technique must be applied to data with the values obtained when training. Moreover, for normalization, 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 when train and test data creation take place, sequences are created during analysis, not after all packets are seen (Figure 6). Thus, each packet can be directly sent to the classification model right after being analyzed.

We let 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 2, 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% ($1/L$) confident with its prediction. Thus 1.5 is selected as the coefficient.

$$probabilityLimit = \frac{1}{L}1.5 \qquad (2)$$

3.5.1. Thread Utilization

A key factor for online traffic classification simulation is packet sender-receiver threads. 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 nonempty. 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.

4. TESTS AND RESULTS

This section describes the dataset used in this work and how it is utilized, test scenarios, and hardware features; test results for LSTM-FS (LSTM with Flow Sequences) are given. Comparisons were made among different studies.

4.1. Dataset

The experiments use the ISCXVPN2016 dataset [10]. 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. [15], 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. [15] has four test scenarios: A-1, A-2 non-VPN, A-2 VPN, and B, as summarized in Table 4. The same scenarios are built for the method LSTM-FS to make meaningful comparisons. In the first part of Scenario A (A-1), the researchers split the dataset into two labels, non-VPN and VPN, to feed into Weka (Figure 7). In the second part of Scenario A (A-2 non-VPN and A-2 VPN), they divide each label's data into six categories and handle both separately (Figure 8 and Figure 9). In Scenario B, this study uses 12 labels (six non-VPN and six VPN categories) to perform classification in one step. All experiment scenarios ran both offline and online for LSTM-FS.

Table 4 Test scenarios

Scenario	Labels
A-1	non-VPN, VPN
A-2 non-VPN	chat, email, file transfer, P2P, streaming, VoIP

A-2 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

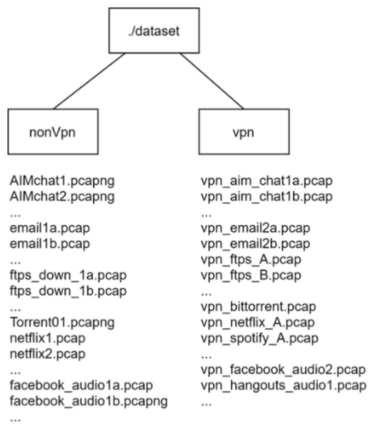

Figure 7 Input directory structure for Scenario A-1

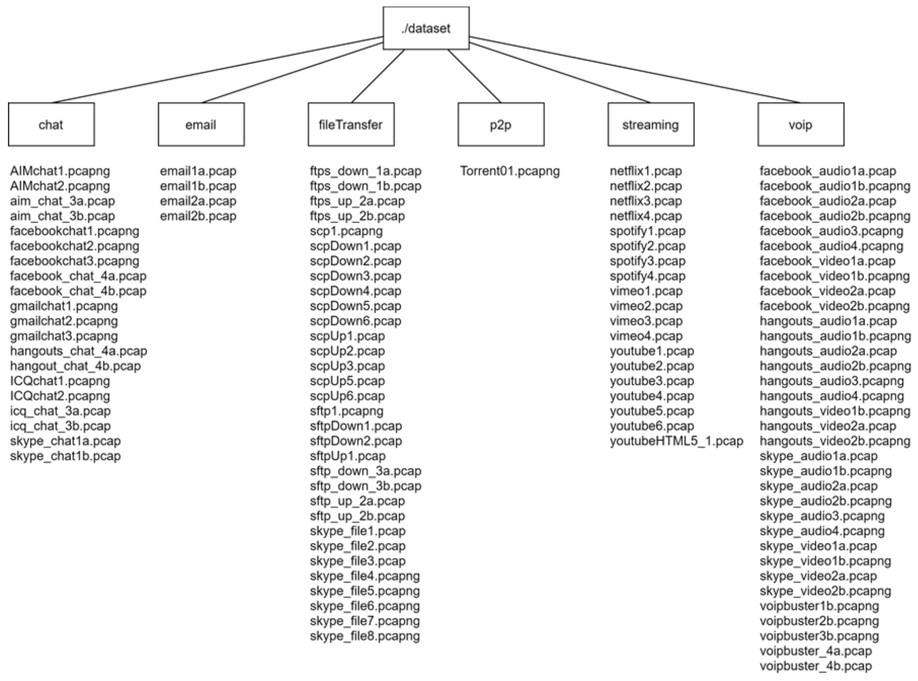

Figure 8 Input directory structure for Scenario A-2 non-VPN

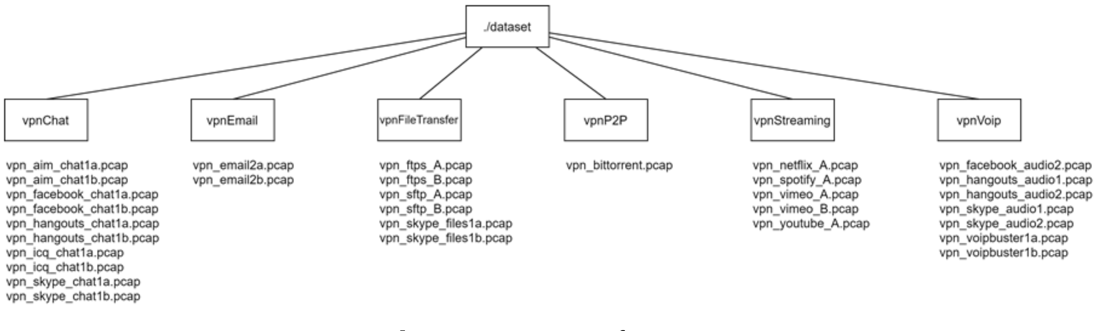

Figure 9 Input directory structure for Scenario A-2 VPN

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

Since it takes too long to process millions of data (See Tables 5, 6, 7), LSTM-FS creates sub-datasets. For every scenario, 10,000 data is chosen randomly per label (Figure 10), in a way that packet order in sequences is preserved. For its tests, it repeats this process five times to generate five models.

Table 5 Scenario A-1 sample distribution before sampling

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

Table 6 Scenario A-2 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 7 Scenario A-2 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

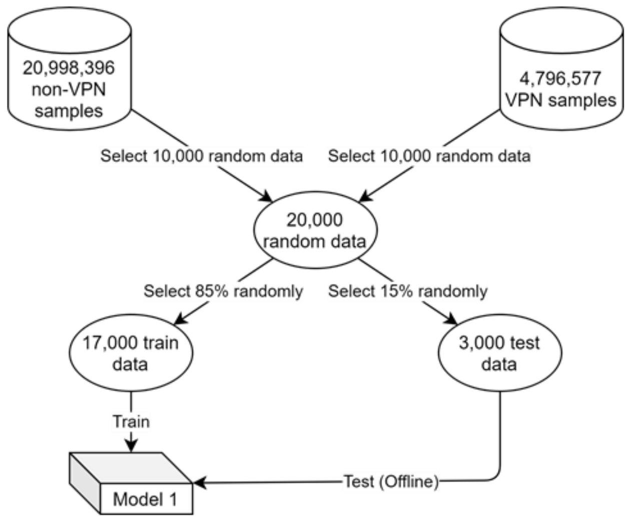

Figure 10 Model generation example (Scenario A-1)

4.3.3. Parameters

All LSTM-FS models use the same 17 features. 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% of the data is used to train, 15% to validate, and 10% 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 A-2 non-VPN and B, Adam learning rate was changed to 0.01 from 0.001. (This had an impact on accuracy around 3%.) 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 Section 3 calculates.

4.4. Test Results

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

4.4.1. Offline Classification

Table 8 exhibits the average accuracy results of offline classification with LSTM-FS for all scenarios. Table 9 shows the average elapsed times.

Table 8 LSTM-FS (Offline) average accuracy results

Scenario	Accuracy	Precision	Recall	F-Score
A-1	99.818	0.9982	0.9982	0.9982
A-2 non-VPN	97.988	0.9804	0.9799	0.9799
A-2 VPN	96.828	0.9691	0.9683	0.9683
B	96.43	0.9658	0.9643	0.9644

Table 9 LSTM-FS (Offline) average elapsed times

Scenario	Data Generation	Training	Testing
A-1	562.73	12.68	2.39
A-2 non-VPN	291.39	25.96	3.54
A-2 VPN	26.62	28.08	3.73
B	432.17	68.29	7.36

4.4.2. Online Classification

After five runs for the first model, as can be seen from Figure 11 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 (1/μ).

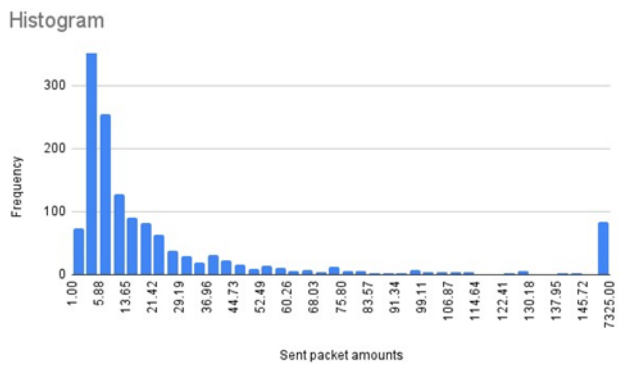

Figure 11 LSTM-FS (Online) Scenario A-1 model 1 sent packet amounts histogram

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.

Table 10 exhibits the average accuracy results of online classification with LSTM-FS for all scenarios. Table 11 shows the average elapsed times.

Table 10 LSTM-FS (Online) average accuracy results

Scenario	Accuracy	Precision	Recall	F-Score	Unknown (%)
A-1	99.8554	0.9986	0.9986	0.9986	6.93
A-2 non-VPN	94.8119	0.9551	0.9515	0.9506	0.65
A-2 VPN	83.8309	0.8477	0.8346	0.8319	5.6
B	88.3211	0.9098	0.8849	0.8874	0.1

Table 11 LSTM-FS (Online) average elapsed times

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

4.5. Evaluation

In the following charts (Figure 12, 13, 14, 15, 16, 17, 18, 19); LSTM-FS (Offline), LSTM-FS (Online) precision and recall values are shown together with the work (C4.5) by Draper-Gil et al. [15]. They did 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.

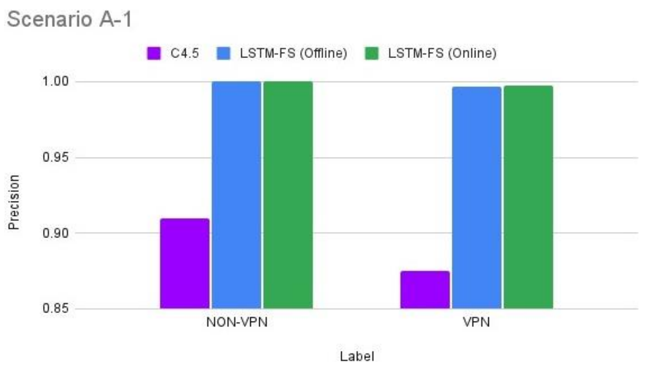

Figure 12 Precision comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario A-1

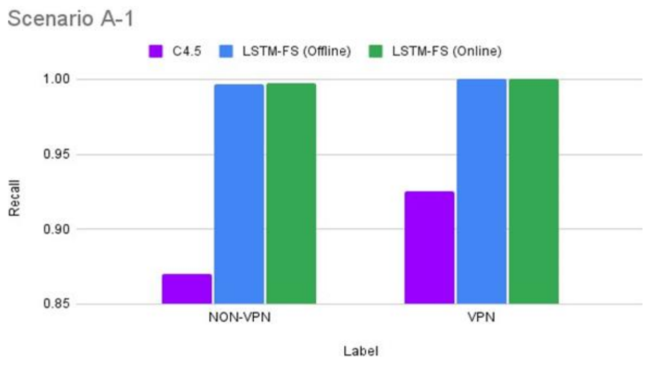

Figure 13 Recall comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario A-1

For scenario A-1, 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.

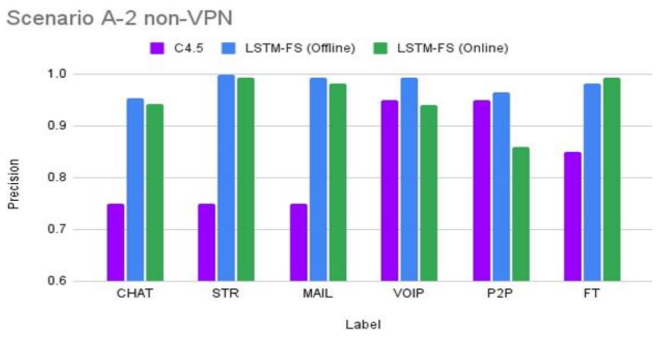

Figure 14 Precision comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario A-2 non-VPN

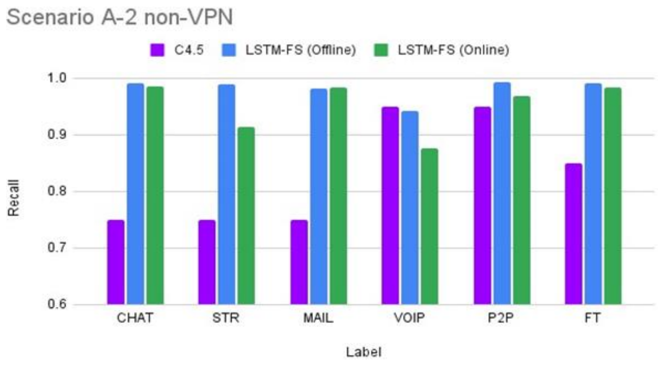

Figure 15 Recall comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario A-2 non-VPN

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.

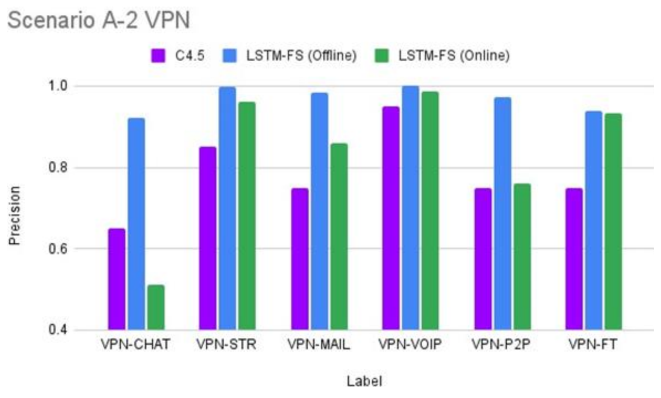

Figure 16 Precision comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario A-2 VPN

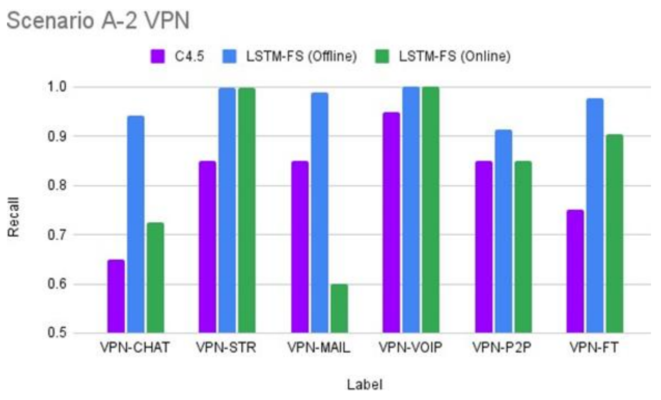

Figure 17 Recall comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario A-2 VPN

Offline LSTM stands out for its Scenario A-2 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 close. VPN-MAIL recall result for online LSTM is especially low.

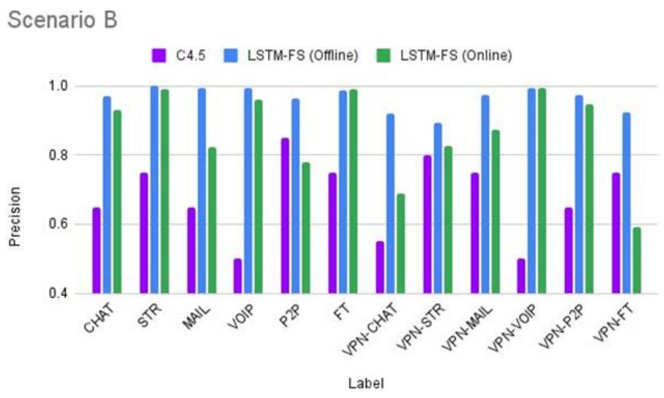

Figure 18 Precision comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario B

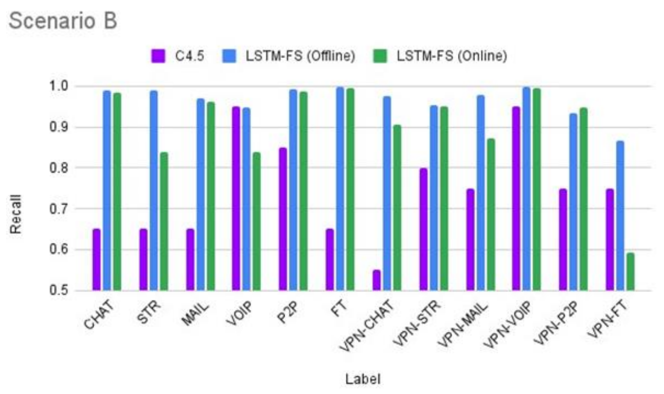

Figure 19 Recall comparison of [15], LSTM-FS (Offline), LSTM-FS (Online) for Scenario B

C4.5 VOIP and VPN-VOIP precision performance are surprisingly not good for Scenario B, even though it excelled for Scenario A-2 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.

Byte Segment Neural Network [27] spends 43 msec 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. Thus, processing overhead per sample is 0.5 msec, since Scenario B consists of 120,000 data. On the other hand, they measured the average time to deal with a sample during online classification as 2.97 msec and shared that for Securitas [28] this is 7.01 msec. Packet processing overhead is at most 0.7477 msec with LSTM-FS for Scenario A-2 non-VPN.

5. CONCLUSION

The problem we tried to attack is to classify encrypted packets using the packet headers. An LSTM deep learning model trained with features that can be taken directly from a network packet is presented in this study. The method yields valid results regarding both its speed and accuracy, outperforming a well-known machine learning study. Compared to the only paper that shared its online classification results [27], the LSTM model presented in this paper gave better results in terms of speed. We are confident that this model can be used for real life situations.

The design 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 algorithm. In addition, a new feature can be easily added to the algorithm, 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 such as intrusion detection systems.

Funding

The authors have not received any financial support for the research, authorship or publication of this study.

The Declaration of Conflict of Interest/Common Interest

No conflict of interest or common interest has been declared by the authors.

The Declaration of Ethics Committee Approval

This study does not require ethics committee permission or any special permission.

The Declaration of Research and Publication Ethics

The authors of the paper declare that they comply with the scientific, ethical and quotation rules of SAUJS in all processes of the paper and that they do not make any falsification on the data collected. In addition, they declare that Sakarya University Journal of Science and its editorial board have no responsibility for any ethical violations that may be encountered, and that this study has not been evaluated in any academic publication environment other than Sakarya University Journal of Science.

Authors' Contribution

The authors contributed equally to the study.

REFERENCES

[1] H. Tahaei, F. Afifi, A. Asemi, F. Zaki and N. B. Anuar, “The rise of traffic classification in IoT networks: A survey,” Journal of Network and Computer Applications, vol. 154, 102538, 2020.

[2] O. Salman, I.H. Elhajj, A. Kayssi et al., “A review on machine learning-based approaches for Internet traffic classification,” Annals of Telecommunications, vol. 75, no. 11, pp. 673-710, 2020.

[3] Z. J. Al-Araji, S. S. S. Ahmad, M. W. Al-Salihi, H. A. Al-Lamy, M. Ahmed, W. Raad and N. M. Yunos, “Network Traffic Classification for Attack Detection Using Big Data Tools: A Review,” Lecture Notes in Networks and Systems, pp. 355-363, 2019.

[4] M. AlSabah, K. Bauer and I. Goldberg, “Enhancing Tor's performance using real-time traffic classification,” ACM conference on Computer and communications security, pp. 73-84, 2012.

[5] IANA. Service Name and Transport Protocol Port Number Registry [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. [Accessed April 27, 2022].

[6] J. Khalife, A. Hajjar and J. Diaz-Verdejo, "A multilevel taxonomy and requirements for an optimal traffic-classification model," International Journal of Network Management, vol. 24, no. 2, pp. 101-120, 2014.

[7] Google. HTTPS encryption on the web [Online]. Available: https://transparencyreport.google.com/https/overview?hl=en. [Accessed April 27, 2022].

[8] S. Rezaei and X. Liu, "Deep Learning for Encrypted Traffic Classification: An Overview," IEEE Communications Magazine. vol. 57. no. 5. pp. 76-81, 2019.

[9] Wireshark [Online]. Available: https://www.wireshark.org. [Accessed April 27, 2022].

[10] UNB. VPN-nonVPN dataset (ISCXVPN2016) [Online]. Available: https://www.unb.ca/cic/datasets/vpn.html. [Accessed April 27, 2022].

[11] B. Yamansavascilar, A. Guvensan, A. Yavuz, and E. Karsligil, "Application identification via network traffic classification," International Conference on Computing, Networking and Communications (ICNC), pp. 843-848, 2017.

[12] E. Hjelmvik. SPID Statistical Protocol Identification [Online]. Available: https://sourceforge.net/projects/spid. [Accessed April 27, 2022].

[13] E. Hjelmvik. The SPID Algorithm Statistical Protocol IDentification. 2008.

[14] E. Hjelmvik and W. John, "Statistical Protocol IDentification with SPID: Preliminary Results," Swedish National Computer Networking Workshop. vol. 9. pp. 4-5, 2009.

[15] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Encrypted and VPN Traffic Using Time-Related Features," In Proceedings of the 2nd international conference on information systems security and privacy (ICISSP), pp. 407-414, Feb. 2016.

[16] S. Bagui, X. Fang, E. Kalaimannan, S. C. Bagui, and J. Sheehan, "Comparison of machine-learning algorithms for classification of VPN network traffic flow using time-related features," Journal of Cyber Security Technology. vol. 1. no. 2. pp. 108-126, 2017.

[17] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural network," IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 43-48, 2017.

[18] echowei. Deep Learning models for network traffic classification [Online]. Available: https://github.com/echowei/DeepTraffic. [Accessed April 27, 2022].

[19] yungshenglu. USTC-TK2016 [Online]. Available: https://github.com/yungshenglu/USTC-TK2016. [Accessed April 27, 2022].

[20] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, "Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning," Soft Computing, vol. 24. no. 3, pp. 1999-2012, 2020.

[21] A. Parchekani, S. N. Naghadeh, and V. Shah-Mansouri, "Classification of Traffic

Using Neural Networks by Rejecting: a Novel Approach in Classifying VPN Traffic," Jan. 2020. arXiv preprint arXiv:2001.03665.

[22] K. Zhou, W. Wang, C. Wu, and T. Hu, "Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks," ETRI Journal, vol. 42, no. 3, pp. 311-323, 2020.

[23] UNB. Tor-nonTor dataset (ISCXTor2016) [Online]. Available: https://www.unb.ca/cic/datasets/tor.html. [Accessed April 27, 2022].

[24] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things," IEEE Access, vol. 5, pp. 18042-18050, 2017.

[25] RedIRIS. Welcome to RedIRIS [Online]. Available: https://www.rediris.es. [Accessed April 27, 2022]

[26] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "nDPI: Open-source high-speed deep packet inspection," International Wireless Communications and Mobile Computing Conference (IWCMC). pp. 617-622, 2014.

[27] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia, "Byte Segment Neural Network for Network Traffic Classification," IEEE/ACM 26th International Symposium on Quality of Service (IWQoS). pp. 1-10, Jun. 2018.

[28] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A Semantics-Aware Approach to the Automated Network Protocol Identification," IEEE/ACM Transactions on Networking. vol. 24. no. 1. pp. 583-595, 2016.

[29] tcpdump. TCPDUMP/LIBPCAP public repository [Online]. Available: https://www.tcpdump.org. [Accessed April 27, 2022].

[30] tcpreplay. tcprewrite [Online]. Available: https://tcpreplay.appneta.com/wiki/tcprewrite. [Accessed April 27, 2022].

[31] Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems [Online]. Available: tensorflow.org. 2015. [Accessed April 27, 2022].

[32] F. Chollet et al. Keras [Online]. Available: https://keras.io. 2015. [Accessed April 27, 2022].

[33] serizba. CppFlow [Online]. Available: https://github.com/serizba/cppflow. [Accessed April 27, 2022].