Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

COUNTERING INTERNET PACKET CLASSIFIERS TO IMPROVE USER ONLINE PRIVACY

by
Sina Fathi-Kazerooni

Internet traffic classification or packet classification is the act of classifying packets using the extracted statistical data from the transmitted packets on a computer network. Internet traffic classification is an essential tool for Internet service providers to manage network traffic, provide users with the intended quality of service (QoS), and perform surveillance. QoS measures prioritize a network's traffic type over other traffic based on preset criteria; for instance, it gives higher priority or bandwidth to video traffic over website browsing traffic. Internet packet classification methods are also used for automated intrusion detection. They analyze incoming traffic patterns and identify malicious packets used for denial of service (DoS) or similar attacks. Internet traffic classification may also be used for website fingerprinting attacks in which an intruder analyzes encrypted traffic of a user to find behavior or usage patterns and infer the user's online activities.

Protecting users' online privacy against traffic classification attacks is the primary motivation of this work. This dissertation shows the effectiveness of machine learning algorithms in identifying user traffic by comparing 11 state-of-art classifiers and proposes three anonymization methods for masking generated user network traffic to counter the Internet packet classifiers. These methods are equalized packet length, equalized packet count, and equalized inter-arrival times of TCP packets. This work compares the results of these anonymization methods to show their effectiveness in reducing machine learning algorithms' performance for traffic classification. The results are validated using newly generated user traffic.

Additionally, a novel model based on a generative adversarial network (GAN) is introduced to automate countering the adversarial traffic classifiers. This model,

which is called GAN tunnel, generates pseudo traffic patterns imitating the distributions of the real traffic generated by actual applications and encapsulates the actual network packets into the generated traffic packets. The GAN tunnel's performance is tested against random forest and extreme gradient boosting (XGBoost) traffic classifiers. These classifiers are shown not being able of detecting the actual source application of data exchanged in the GAN tunnel in the tested scenarios in this thesis.

COUNTERING INTERNET PACKET CLASSIFIERS TO IMPROVE USER ONLINE PRIVACY

by
Sina Fathi-Kazerooni

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

Helen and John C. Hartmann Department of Electrical and Computer Engineering

December 2020

APPROVAL PAGE

COUNTERING INTERNET PACKET CLASSIFIERS TO IMPROVE USER ONLINE PRIVACY

Sina Fathi-Kazerooni

Dr. Roberto Rojas-Cessa, Dissertation Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Senjuti Basu Roy, Committee Member — Date
Assistant Professor of Computer Science, NJIT

Dr. Sui-Hoi Edwin Hou, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Qing Liu, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Dr. Bipin Rajendran, Committee Member — Date
Reader in Engineering, King's College London

BIOGRAPHICAL SKETCH

Author: Sina Fathi-Kazerooni

Degree: Doctor of Philosophy

Date: December 2020

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2020
- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2016
- Bachelor of Science in Electrical Engineering,
  Shiraz University, Shiraz, Iran, 2014

Major: Electrical Engineering

Presentations and Publications:

S. Fathi-Kazerooni, R. Rojas-Cessa, Z. Dong, and V. Umpaichitra, "Correlation of Subway Turnstile Entries and COVID-19 Prevalence and Deaths in New York City," *Elsevier Infectious Disease Modelling*, pp. 1-11, Nov 2020. (In Press)

S. Fathi-Kazerooni and R. Rojas-Cessa, "GAN Tunnel: Network Traffic Steganography by using GANs to Counter Internet Traffic Classifiers," *IEEE Access*, vol. 8, pp. 125345 - 125359, July 2020.

S. Fathi-Kazerooni, Y. Kaymak, and R. Rojas-Cessa, "Identification of User Application by an External Eavesdropper using Machine Learning Analysis on Network Traffic," *IEEE International Conference on Communications Workshops (ICC Workshops)*, Shanghai, China, 2019, pp. 1-6

S. Fathi-Kazerooni, Y. Kaymak, and R. Rojas-Cessa, "Tracking User Application Activity by using Machine Learning Techniques on Network Traffic," *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Okinawa, Japan, 2019, pp. 405-410

Y. Kaymak, S. Fathi-Kazerooni, and R. Rojas-Cessa, "Indirect Diffused Light Free-space Optical Communications for Vehicular Networks," *IEEE Communications Letters*, vol. 23, no. 5, pp. 814-817, May 2019.

A. Agnihotri, S. Fathi-Kazerooni, Y. Kaymak, and R. Rojas-Cessa, "Evacuating Routes in Indoor-Fire Scenarios with Selection of Safe Exits on Known and Unknown Buildings using Machine Learning," *IEEE 39th Sarnoff Symposium*, Newark, NJ, USA, 2018, pp. 1-6

S. Fathi-Kazerooni, Y. Kaymak, R. Rojas-Cessa, J. Feng, N. Ansari, M. Zhou, and T. Zhang, "Optimal Positioning of Ground Base Stations in Free-space Optical Communications for High-speed Trains," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1940–1949, June 2018

S. Fathi-Kazerooni and R. Rojas-Cessa, "SRA: Slot Reservation Announcement Scheme for Medium Access Control of IEEE 802.11 Crowded Networks in Emergency Scenarios," *IEEE International Conference on Communications (ICC)*, Paris, 2017, pp. 1-6

تقدیم به پدر و مادر عزیزم

ACKNOWLEDGMENT

I would like to express my gratitude to my dissertation advisor, Dr. Roberto Rojas-Cessa, who has been actively helping me toward the completion of this work.

I also like to thank Dr. Yagiz Kaymak and Dr. Ziqian Dong for all their help with our various research projects.

I would like to also thank all my dissertation committee members Dr. Senjuti Basu Roy, Dr. Sui-Hoi Edwin Hou, Dr. Qing Liu, and Dr. Bipin Rajendran for their participation as the defense committee and also their helpful comments.

I would like to thank Helen and John C. Hartmann Department of Electrical and Computer Engineering for supporting me financially through parts of my doctoral program.

Finally, I like to thank Ms. Gonzalez-Lenahan and Dr. Ziavras who helped me to revise this work.

TABLE OF CONTENTS

Chapter **Page**

1 INTRODUCTION 1

1.1 Internet Traffic Classifiers 1

1.2 Machine Learning Models used for Traffic Classification 4

1.2.1 Decision Tree 5

1.2.2 Adaboost 6

1.2.3 XGBoost 7

1.2.4 Bagged Trees 8

1.2.5 Random Forest 8

1.2.6 Naive Bayes 9

1.2.7 Stochastic Gradient Descent 10

1.2.8 Support Vector Machines 10

1.2.9 Logistic Regression 11

1.2.10 Multi-layer Perceptron 11

1.2.11 Generative Adversarial Networks 12

1.2.12 Forecasting Models 15

1.2.13 Kolmogorov-Smirnov Two Sample Test 17

1.2.14 Performance Metrics 18

1.3 Countering Internet Traffic Classifiers 19

1.4 Traffic Statistics Equalization 20

1.5 GAN Tunnel 20

1.6 Summary of Contributions 21

2 TRACKING USER APPLICATION ACTIVITY BY USING MACHINE LEARNING TECHNIQUES ON NETWORK TRAFFIC 22

2.1 Online Activity Tracking 23

2.2 Classification Evaluation 28

2.3 Conclusion 32

TABLE OF CONTENTS
(Continued)

Chapter **Page**

3 COUNTERING INTERNET PACKET CLASSIFIERS TO IMPROVE USERS ONLINE PRIVACY 36

3.1 Background and Related works 36

3.1.1 Labeling Techniques 36

3.1.2 Countermeasures against ITCs 37

3.2 System Description 37

3.2.1 Dataset Details 38

3.2.2 Masking Methods 39

3.3 Evaluation Results 41

3.3.1 Split Test Dataset 43

3.3.2 Unknown Test Dataset 48

3.4 Discussion 54

3.4.1 Impact of Traffic Statistics Modifications 54

3.4.2 Comparison with Related Works 58

3.5 Conclusion 59

4 GAN TUNNEL: NETWORK TRAFFIC STEGANOGRAPHY BY USING GENERATIVE ADVERSARIAL NEURAL NETWORKS TO COUNTER INTERNET TRAFFIC CLASSIFIERS 64

4.1 Model Description 65

4.1.1 Dataset Information and Preprocessing 66

4.1.2 Flow Generation by Using WGAN 67

4.1.3 Packet Steganography and GAN Tunnel 70

4.2 Performance Evaluation 70

4.2.1 Evaluation of Implemented WGANs 71

4.2.2 Time Complexity and Run Time 74

4.2.3 Evaluation of WGANs and ITCs Trained on the Same Dataset 74

4.2.4 Evaluation of WGANs and ITCs Trained on Different Datasets 77

4.2.5 Parameter Tuning of WGAN 79

TABLE OF CONTENTS
(Continued)

Chapter **Page**

4.2.6 Packet Generation from WGAN Flows 79
4.3 Background and Related works 83
4.4 Conclusion 89
5 CONCLUSION 91
REFERENCES 93

LIST OF TABLES

Table		Page
2.1	Applications and their Categories	29
2.2	Results of the Split-train-test Dataset Evaluation	30
2.3	Results of the Unknown Traffic Evaluation	33
3.1	Dataset Information.	40
3.2	Kolmogorov-Smirnov Test: Two-tailed P-values of Applications Packet Counts	43
3.3	F1-score Distributions Stats on the Split Dataset.	47
3.4	F1-score distributions stats on the unknown dataset	52
3.5	Countermeasures against ITCs	63
4.1	Training Dataset Information before Balancing	67
4.2	WGAN Design	69
4.3	Run Time to Generate 1,000 Flows	75
4.4	RF Results on Generated Data	76
4.5	XGBoost Results on Generated Data	77
4.6	RF Results on Generated Data by using Split Dataset	78
4.7	XGBoost Results on Generated Data by using Split Dataset	78
4.8	Range of Parameters of WGANs Designs	79
4.9	WGANs With Different Parameters	80
4.10	A sample of a synthetic flow of Google Drive	82
4.11	Overview of Packets Needed for Handshakes and Data in a Generated Flow by our WGAN without using TLS	83
4.12	Overview of Packets Needed for Handshakes and Data in a Generated Flow by our WGAN with TLS	83
4.13	Comparison of objectives of security systems employing GAN	85
4.14	Comparison between architecture of security systems employing GAN	86

LIST OF FIGURES

Figure		Page
2.1	Random Forest for traffic classification.	24
2.2	Representation of application identification made by an attacker external to a user's network.	25
2.3	Detailed classification process.	27
2.4	Graphical representation of true positives, false negatives, true negatives, and false positives.	28
2.5	Precision, recall, and F1-score in our classifier for each application in the split-train-test dataset evaluation.	31
2.6	Normalized confusion matrix of the classified applications in the split-train-test dataset evaluation.	32
2.7	Precision, recall, and F1-score in our classifier for each application in the unknown traffic evaluation.	33
2.8	Normalized confusion matrix of the classified applications in the unknown traffic evaluation.	34
2.9	Precision, recall, and F1-score in our classifier for each application in the two-application traffic evaluation.	35
2.10	Normalized confusion matrix of the classified applications in the two-application traffic evaluation.	35
3.1	System model.	37
3.2	Empirical CDF lengths of packets generated by different applications.	41
3.3	Empirical CDF of counts of packets generated by different applications.	42
3.4	Empirical CDF of inter-arrival times of packets generated by different applications.	44
3.5	Macro-averaged precision of different ML algorithms on the split test dataset of original data	45
3.6	Macro-averaged precision of different ML algorithms on the split test dataset of equalized packet length	46
3.7	Macro-averaged precision of different ML algorithms on the split test dataset of equalized packet count	47
3.8	Macro-averaged precision of different ML algorithms on the split test dataset of equalized inter-arrival times	48

LIST OF FIGURES
(Continued)

Figure **Page**

3.9 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of original data. 49

3.10 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of equalized packet length. 50

3.11 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of equalized packet count. 51

3.12 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of equalized packet inter-arrival times. 52

3.13 F1-score cumulative KDE of the masking methods on the split dataset. . 53

3.14 Macro-averaged precision of different ML algorithms on the unknown dataset of original data. 54

3.15 Macro-averaged precision of different ML algorithms on the unknown dataset of equalized packet length. 54

3.16 Macro-averaged precision of different ML algorithms on the unknown dataset of equalized packet count. 55

3.17 Macro-averaged precision of different ML algorithms on the unknown dataset of equalized inter-arrival times. 55

3.18 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on original data. 56

3.19 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on equalized packet length. 57

3.20 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on equalized packet count. 58

3.21 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on equalized packet inter-arrival times. . . . 59

3.22 F1-score cumulative KDE of the masking methods on the unknown dataset for all three methods. 60

3.23 Confusion matrices. 61

4.1 WGAN tunnel system overview. 66

4.2 WGAN architecture adapted for network traffic generation. 66

4.3 Sequence of Processes in the GAN Tunnel Client/Server. 68

4.4 Wasserstein-1 values per application. 71

LIST OF FIGURES
(Continued)

Figure **Page**

4.5 Kernel density estimations of generated and actual traffics' features for each application. 72

4.6 F-1 scores of RF on different WGANs. WGAN 6 achieves the highest F-1 score 81

CHAPTER 1

INTRODUCTION

1.1 Internet Traffic Classifiers

Internet traffic classifiers (ITCs) use the statistical properties of the transmitted packets on a network to classify them and infer information from them [1,2]. Internet service providers (ISPs) use ITCs to manage and monitor their network traffic and provision the quality of service (QoS) [3–5]. QoS-provisioning mechanisms secure service guarantees to flows as required by a service level agreement. For instance, ISPs would be required to preserve and support multimedia traffic despite the presence of other traffic in the network [1, 2].

ITCs can identify visited websites, services, or applications that generate network traffic. Such information is used to aid ISPs' operations. ITCs may also be used to detect adversarial attacks, such as network intrusion and distributed denial of service (DDoS) [6,7]. Nevertheless, traffic classification may also be used for adversarial purposes, such as invading user privacy [8, 9]. Therefore, ITCs may also be considered as a threat to confidentiality [10]. Moreover, ITCs may also be used to perform Internet censorship [11].

There are three groups of Internet traffic classification methods [12]: payload-based, port-based, and statistical-based. Payload-based method uses application signatures in payload data for deep packet inspection and classification [13]. The port-based method identifies packets based on the Internet Assigned Numbers Authority (IANA) list of registered applications and their assigned port numbers. The statistical method uses packet features such as size and inter-arrival times to detect the pattern of exchanged packets for classification.

Port identification might be fruitless for traffic classification as different applications may use the same well-known port numbers, such as port 443, listed by

IANA [14], and such number is used by both secure sockets layer (SSL) and virtual private networks (VPNs). Therefore, port-number detection has become ambiguous in this application [15, 16]. Moreover, adoption of encrypted communications by various applications is on the rise, and therefore, ITCs cannot rely on encrypted packet payloads for classification [12,17]. Instead, ITCs use traffic statistics, including lengths and timing of packets and transport layer information, such as transmission control protocol (TCP) window sizes for classification [3, 5]. Nguyen et al. [5] list the features used by preliminary works for traffic classification. Based on this list, we use the frequently used features that are extracted from TCP headers. These features include flow size, packet interarrival times, packet counts of flows, and TCP window sizes.

ITCs primarily use machine learning (ML) algorithms, such as Random Forest [6–9, 18–20]. These ITCs mainly focus on traffic classes such as web browsing, email, P2P, streaming, gaming, and file transfer. However, more specific information, such as users' software, is also targeted and it may be used to detect user online activity [2].

There are several recent studies on Internet packet classification [3–9, 12, 13, 15, 16, 18, 18–26]. One of the main challenges tackled by these studies is labeling the collected traffic traces for classification. Here, labeling refers to the association of the application generating a packet and the packet itself. Labeling requires access to the user's computational resources for establishing ground-truth properties and information for algorithm training. However, such access may raise privacy concerns for who, despite acquiring a benefit, allow such access for information disclosure. Previous studies label the traffic traces by using well-known port numbers. These works match each captured packet to a service using the port numbers listed by IANA [3, 4, 23]. This labeling methodology limits the classification to group classification because many applications use the same well-known port numbers, such as port 80 by HTTP. For instance, Google Drive and Microsoft OneNote packets carry port 443 as the destination port number to connect to their respective servers.

Li et al. [21] used Naive Bayes, Decision Tree, and Adaboost for traffic classification and achieved up to 99% accuracy. The classification of this method is limited to identifying application classes. The flows in their dataset are reported to be hand-classified using a content-based mechanism into 10 classes of applications.

Sen et al. [27] used traffic analysis to identify P2P traffic using application signatures. Similarly, Yu et al. [13] proposed to use expression matching on packet payloads for traffic classification. Application signatures are extracted from HTTP request headers that are included in packet payloads and sometimes from TCP headers. The main drawback of this work is the encryption applied to most packet payloads nowadays.

Yamansavascilar et al. [28] used a labeling method to capture the packets by using only one application at a time and labeled them accordingly, and then combined the captured files to construct a dataset. This method, however, limits the amount of the captured data as it requires direct intervention by the user.

Draper-Gil et al. [24] and Lotfollahi [7] used a combination of regular traffic and VPN traffic for classes of applications. Each class of traffic was captured separately for labeling. This labeling method is also limited because of the use of both classes of applications and the requirement of capturing one class at a time.

Clustering methods are also employed for Internet traffic classification [23, 29, 30]. These types of classification identify similar flows and group them together in one cluster. These clusters are identified by port mapping as different classes of traffic, such as DNS and HTTP [23].

Wang et al. [31] used a one-dimensional convolutional neural network (CNN) to detect traffic type groups such as email, chat, streaming, and file transfer. CNNs are a good choice for byte-level traffic classification because they are mainly used in image classification, where the input data are formed from vectors of RGB values for each image pixel [32]. In the encrypted traffic classification experiment, they achieve about 86% accuracy, 84.3% precision, and 79.3% recall. Taylor et al. [33] study the

change in mobile application fingerprints and their detectability over time. Aceto et al. [34,35] study mobile application identification using stacked auto encoders, CNN, and long short-term memory. They did not consider countering ITCs in their study. SAEs are used to learn unsupervised feature learning and recreate the actual data at the output [34]. By adding MLP layers at the output of SAE, the model is capable of classifying network traffic. LSTMs are mainly used to model time-series data, where the model benefits from both historical and recent data. In other words, LSTM aims to learn the time evolution of sequences of data [32].

In this study, we use a modified version of Wireshark packet sniffer [36], called Wireshark PAINT [37], which uses event tracing for Microsoft Windows (ETW) [38] to capture and label traffic. ETW associates the generated network traffic with the issuing process identification (ID), and in this way, it enables Wireshark PAINT to identify the originating application by matching the process ID with process name. For example, the packets generated by Google Chrome are labeled as "chrome.exe".

1.2 Machine Learning Models used for Traffic Classification

In this research, we employ various ML models to show the effectiveness of ITCs. We use two boosting methods; Adaboost and XGBoost. Boosting methods combine a number of weak classifiers, such as small decision trees, to create a powerful classifier [39]. We also use bagged trees, in which decision trees are used as classifiers on a randomly selected subset of the original data. After that, the final result is obtained by performing either averaging or voting [40]. Bagging reduces the high variance of decision-tree classifiers to reduce the classification error and to improve the final classification results [39]. We also use two Naive-Bayes (NB) classifiers, namely, Bernoulli NB and Gaussian NB. The NB classifiers work under the assumption that data features are independent (naive assumption) and have Gaussian and Bernoulli distributions. These distributions apply for the Gaussian- and Bernoulli-NB classifiers, respectively. The Stochastic Gradient Descent (SGD)

classifier is an estimation of gradient descent optimization [41], in which the gradient of the objective function is estimated rather than calculated. The Support Vector Machine (SVM) classifier constructs a hyperplane to separate different classes of data in a multi-dimensional space [39]. The Multi-layer Perceptron (MLP) classifier is a neural network (NN) model with one or more hidden NN layers. The dataset features are inputs to the NN, and the output is the classification result [40]. The voting classifier comprises different classifiers and calculates its final prediction by voting based on the results of the implemented classifiers. We consider Logistic Regression, Random Forest, and Gaussian NB for the voting classifier. Logistic Regression is a classification method that uses a linear function to calculate the likelihood of an output class or that an application may have generated a specific TCP packet [39]. Random Forest consists of a stack of decision-tree predictors [42]. These decision-trees are trained on a dataset and their results are averaged as the final classification result [40].

1.2.1 Decision Tree

Classification and regression trees (CART) or simply Decision Tree are binary trees [43], where each tree node represents a feature, and each leaf represents the prediction. The division of the input space recursively creates a Decision Tree. To perform the required splitting, all input features are evaluated to find the best feature for each split. The dataset is then split into subsets based on the selected best feature. This recursive process creates new tree nodes and splits them until it achieves the desired classification accuracy versus the number of nodes. The classification accuracy is defined as number of correctly classified instances divided by total number of instances. The selection of best feature to split the dataset uses the Gini index function, $\sum \hat{p}_{mk}(1 - \hat{p}_{mk})$, where $\hat{p}_{mk}$ is the probability of classifying data into class k [39].

1.2.2 Adaboost

Adaboost classifier first trains a weak classifier, a classifier that performs poorly but better than random guessing, for instance, a Decision Tree. The base model at first assumes equal weights for each sample in the training data, x_i. A sample weight indicates how important it is to correctly classify that sample. This base classifier makes predictions on the training set. Afterward, Adaboost increases the relative classification weights of the misclassified training instances. Adaboost then trains a second weak classifier using the updated weights. The second classifier makes predictions on the training set and updates the weights of training samples, and gives more weights to incorrectly classified samples based on predictions result compared to actual classes (labels) of samples. Therefore, each subsequent weak classifier focuses more on correctly classifying the previously misclassified samples. This process continues until a predefined number of base classifiers are trained [40,44].

With m samples in the training data, initially, the weight of i^{th} sample, $w^{(i)}$, is set to $\frac{1}{m}$. The first classifier computes the weighted error, r_1, on the training data. The weighted error for j^{th} classifier is calculated as

$$r_j = \frac{\sum\limits_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\sum\limits_{i=1}^{m} w^{(i)}}, \tag{1.1}$$

where $\hat{y}_j^{(i)}$ is the j^{th} weak classifier prediction of i^{th} training sample [44]. The first classifier's weight, α_j, is then computed as

$$\alpha_j = \log \frac{1 - r_j}{r_j}. \tag{1.2}$$

The classifier's weight is higher, the more accurate that classifier's predictions are [44]. The training samples weights are updated as:

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases} \tag{1.3}$$

for $i = 1, 2, ..., m$. The instance weights are normalized by being divided by $\sum_{i=1}^{m} w^{(i)}$. The next weak classifier is trained using the updated weights [44]. The final prediction of the Adaboost model is

$$\hat{y}(x) = \arg\max_{k} \sum_{\substack{j=1 \\ \hat{y}_j^{(x)}=k}}^{N} \alpha_j, \tag{1.4}$$

where N is the number of weak classifiers [44].

1.2.3 XGBoost

XGBoost is a boosting method similar to Adaboost. The main difference between the two is that XGBoost uses constant training sample weights. Instead, XGBoost fits a weak classifier to the residual errors made by the previous classifier [44]. Let us consider the residual errors in form of mean squared error, $L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$, where i is the index of training samples. We initialize the model with a constant value γ as

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y^{(i)}, \gamma). \tag{1.5}$$

For each subsequent weak classifier, j with $j = 1, 2, ..., N$, the model computes

$$r_j^{(i)} = -\left[\frac{\partial L(y^{(i)}, f(x_i))}{\partial f(x_i)}\right]_{f=f_{j-1}}. \tag{1.6}$$

Then, the model fits the weak classifier to $r_j^{(i)}$ [39]. For weak classifier j, we compute γ_j as

$$\gamma_j = \arg\min_{\gamma} \sum_{i=1}^{m} L(y^{(i)}, f_{j-1}(x_i) + \gamma y_j(x_i)). \tag{1.7}$$

The model is updated by using

$$f_j(x) = f_{j-1}(x) + \gamma_j y_j(x). \tag{1.8}$$

After training all weak learners, the model output is f_N [39].

1.2.4 Bagged Trees

The bootstrap averaging, or bagging, improves a weak classifier's performance, such as a Decision Tree, by reducing the variance of predictions through averaging [39]. For example, for a set of n independent predictions, each with a variance of σ^2, the variance of averaged predictions is $\frac{\sigma^2}{n}$. Therefore, to improve the performance of Decision Tree, Bagged-Trees classifier trains a Decision-Trees classifier on aggregated subsets of training data with replacement and produces a final classification result.

1.2.5 Random Forest

Similar to Bagged Trees, Random Forest (RF) calculates the final predictions based on the results of weak classifiers (for instance, Decision Tree) [42]. The main difference between Bagged Trees and RF is the subset selection of the training dataset. In RF,

a subset of data is chosen at random, including a portion of features. This subset is then used to train a Decision-Tree classifier. This process is repeated on other random subsets of training data, and the final prediction is given based on the mode of trained Decision-Tree classifiers' predictions.

1.2.6 Naive Bayes

Naive Bayes (NB) classifier is based on Bayesian probability function, $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, where $P(A)$ and $P(B)$ are the probabilities of events A and B, and $P(A|B)$ is the conditional probability of event A given the probability of event B is true [45]. To use Bayes' theorem for classification, we calculate maximum $P(A_i|B)$ where A_i is a feature of training dataset and B is a class of it, $max(P(A_i|B)) = \frac{max(P(B|A_i)P(A_i))}{P(B)}$. For a dataset with numerous features, calculating the joint probability, $P(B|A_i)P(A_i)$, is impractical. To overcome this problem, we use a naive assumption that features are not correlated with each other. Therefore, $P(B|A_i)$ is calculated as $P(B|A_1)P(B|A_2)...P(B|A_m)$, where m is the number of considered features. The Bernoulli NB classifier implements NB classification for data that have a multivariate Bernoulli distribution. Similarly, in Gaussian NB, the likelihood of the features is assumed to follow a Gaussian distribution.

The network traffic data considered in this thesis includes several continuous features, such as packet interarrival times, TCP window sizes, and length of packets. In this work, we also use the number of packets per TCP flow, a discrete feature. Naive Bayes classifier is optimal for data with nominal features [46], and therefore, it is not suitable to be used with network traffic data without modifying the dataset. Because most of our dataset features are continuous, we use a Gaussian NB classifier. Such a classifier considers features with a normal distribution and fits the dataset with mostly continuous features. Bernoulli NB uses features as having Bernoulli distribution [47]. In a Bernoulli distribution, each feature is converted to a binary format. For example,

if the interarrival times are larger than a constant preset threshold (e.g., zero), we consider it to be one. Otherwise, we assign zero to it.

1.2.7 Stochastic Gradient Descent

The stochastic gradient descent (SGD) classifier is based on the gradient descent optimization method [41], in which the rate of change of a dependent variable is calculated as the degree of change of the independent variables. SGD is applicable when calculating the gradient descent of a complex set of training data is infeasible. SGD is an iterative method aiming at finding the parameters of the prediction function by minimizing a cost function. SGD randomly chooses samples from the dataset to calculate the local minima of the cost function.

1.2.8 Support Vector Machines

SVM algorithm performs classification by finding hyperplanes that maximize the margin between features data points [39]. The vectors that define the hyperplanes are called support vectors. Let us assume that the training data, x_i, is given in two classes, $y \in \{1, -1\}$. The goal of SVM classifier is to find vectors w and b that define the target hyperplane, such that $sgn(w^T\phi(x_i) + b)$ returns the correct class (label) for majority of x_i [40]. The sign function, sgn, is defined as

$$sgn(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } = 0, \\ 1 & \text{if } x \geq 0. \end{cases} \tag{1.9}$$

The classification problem is formulated as [40]:

$$\begin{aligned} \underset{w,b,\zeta}{\text{minimize}} \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{m}\zeta_i \\ \text{subject to} \quad & y^{(i)}(w^T\phi(x_i)+b) \geq 1-\zeta_i, \\ & \zeta_i \geq 0, \\ & i = 1, ..., m. \end{aligned}$$

Here, ζ_i creates a margin around the hyperplane that allows some data points to be misclassified if they are in the margin. This margin is called a soft margin. C is a constant number that controls the soft margin penalty size. A larger C allows more misclassified points around the hyperplane.

1.2.9 Logistic Regression

Logistic regression (or logit regression) is a classification method that estimates the probability of a data point belonging to a particular class [44]. The logistic function is defined as $\sigma(t) = \frac{1}{1+exp(-t)}$ [44]. The estimated probability is calculated as $\hat{p} = h_\theta(x) = \sigma(x^T\theta)$, where $x^T\theta$ is a linear regression fitted on the training data. The model prediction is calculated as

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5, \\ 1 & \text{if } \hat{p} \geq 0.5. \end{cases} \tag{1.10}$$

1.2.10 Multi-layer Perceptron

Multi-layer Perceptron (MLP) is a machine learning model that learns a function $f(.) : R^D \rightarrow R^o$, where D is the number of features and o is the number of output classes [40]. Considering that the leftmost layer of MLP is the input layer, the rightmost

layer is then the output layer. The layers between input and output layers are called hidden layers. Let us assume $d = 1, 2, ..., D$, then, the input layer of MLP consists of D units (or neurons), x_d. Each hidden layer unit transforms values from previous layer with a weighted linear summation, $\sum_{d=1}^{D} w_d x_d$. This weighted average is then transformed by an activation function [40]. The activation function is used to add non-linearity to layer outputs. The common activation functions include hyperbolic tangent, $tanh(.)$, rectified linear unit (ReLU), and Leaky ReLU. The ReLU assigns a zero gradient to neural network units with negative or zero inputs, and therefore, the units are deactivated with such inputs. Simply put, for an input value of x, the ReLU outputs $max(x, 0)$. Unlike ReLU, the Leaky ReLU allows adding a small gradient to the units when they are inactive. That gradient improves the performance of the neural network by increasing sparsity and dispersion of hidden units activation [52]. The Leaky ReLU's output for an input value of x is αx for $x < 0$ and x otherwise, where α is a small slope coefficient for negative inputs.

1.2.11 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are originally used to generate fake images that are indistinguishable from real ones [53]. Similar to generating images, GANs may be used to model a statistical distribution. To model a target distribution, $\mathbb{P}_{\mathrm{r}}$, first, we define a parametric family of densities $(P_g), g \in \mathbb{R}^d$ [54]. Then, we choose the density that maximizes the likelihood on real data as the target distribution model. With real data samples set of $\left\{x^{(i)}\right\}_{i=1}^{m}$, the target distribution is calculated by solving [54]:

$$\max_{g \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \log P_g(x^{(i)}). \tag{1.11}$$

GANs model a target distribution by training two competing neural networks, namely a generator and a discriminator. The goal of a generator is to output data samples

that closely follow $\mathbb{P}_{\mathrm{r}}$. The generator uses a random variable z with a distribution $p(z)$, such as a uniform distribution, and inputs and transform z through a neural network that generates samples with a distribution $\mathbb{P}_{\mathrm{g}}$. The uniform and normal distributions are the common practice for input noise distributions of GANs and it has been shown that they lead to similar results [53–57].

$\mathbb{P}_{\mathrm{r}}$ and $\mathbb{P}_{\mathrm{g}}$ are the distributions of real and generated data, respectively. The goal of the discriminator is to check if $\mathbb{P}_{\mathrm{g}}$ is close to $\mathbb{P}_{\mathrm{r}}$ by using a loss function, such as the cross-entropy [58]. If the discriminator can detect the generated samples from the actual ones, the generator updates its neural network model to improve the generated samples. This competition between the generator and discriminator is a *minimax* game-theory optimization [53]:

$$\begin{aligned} &\min_G \max_D V(D,G) \\ &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] \\ &+\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))], \end{aligned} \tag{1.12}$$

where V is the value function, $D(x)$ is the probability calculated by discriminator that x is from $\mathbb{P}_{\mathrm{r}}$, and $G(z)$ is the generator mapping from the noise distribution to $\mathbb{P}_{\mathrm{g}}$. Here, the generator produces synthetic data from a uniform distribution. Meanwhile, the discriminator learns to differentiate generated and real samples with higher accuracy. The discriminator increasingly fails to distinguish the genuine samples from the synthetic ones as the training of the GAN progresses and the generator improves its modeling of the traffic distribution. At some point, the generator samples are satisfactory and the discriminator is no longer needed.

An version of GAN is called Wasserstein GAN (WGAN) [54]. The key advantage of a WGAN over a GAN is the correlation of its loss function to the quality of generated data [54]. A random noise is used as the input to the generator. The

generator outputs the pseudo traffic (generated traffic). This generated traffic and actual network traffic are the two inputs of the critic. The critic then evaluates the similarity level between generated and actual traffic. WGAN differs from GAN in the way it calculates the difference between synthetic and real data distributions; the loss function in GAN is standard cross-entropy, but in WGAN, the loss function is the Earth-Mover (EM) distance, or Wasserstein-1 [53, 54]. Additionally, in WGAN, the discriminator is called the critic, and it estimates the EM distance between the distributions of actual data and the generated ones, as [54]:

$$W(\mathbb{P}_{\mathrm{r}}, \mathbb{P}_{\mathrm{g}}) = \inf_{\gamma \in \prod(\mathbb{P}_{\mathrm{r}}, \mathbb{P}_{\mathrm{g}})} \mathbb{E}_{(\mathrm{x},\mathrm{y})\sim\gamma} \left[\|\mathrm{x} - \mathrm{y}\| \right]. \tag{1.13}$$

Here, $\prod(\mathbb{P}_{\mathrm{r}}, \mathbb{P}_{\mathrm{g}})$ is the set of all joint distributions, $\gamma(x, y)$, whose marginal distributions are $\mathbb{P}_{\mathrm{r}}$ and $\mathbb{P}_{\mathrm{g}}$. In other words, the similarity between real and generated data is calculated by finding the infimum of the expected values of distances between data points from the distributions of real and generated data.

Another version of WGAN is WGAN with gradient penalty (WGAN-GP) [59]. To minimize the divergence in a WGAN, one requirement is that the discriminator value function (Wasserstein-1) be continuous, and therefore, differentiable. To satisfy this requirement, WGAN enforces the discriminator to lie within a certain space (1-Lipschitz) by using weight clipping. On some occasions, this leads to poorly constructed images by generator [59]. A 1-Lipschitz function is differentiable if and only if its gradients norm is equal to 1 everywhere [59]. To overcome the WGAN's undesirable behavior, WGAN-GP enforces the 1-Lipschitz constraint by directly penalizing the gradient norm. The loss function in WGAN-GP is

$$L = \mathop{\mathbb{E}}_{\widetilde{x}\sim\mathbb{P}_g} [D(\widetilde{x})] - \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r} [D(x)] + \lambda \mathop{\mathbb{E}}_{\widehat{x}\sim\mathbb{P}_{\widehat{x}}} \left[\left(\|\nabla_{\widehat{x}} D(\widehat{x})\|_2 - 1 \right)^2 \right], \tag{1.14}$$

where $\lambda \underset{\widehat{x}\sim\mathbb{P}_{\widehat{x}}}{\mathbb{E}} \left[\left(\left\| \nabla_{\widehat{x}} D(\widehat{x}) \right\|_2 - 1 \right)^2 \right]$ is the gradient penalty [59].

Another interesting GAN model is bidirectional GAN (BiGAN) [60]. GANs map latent space (such as a noise distribution) samples to generated data. The latent space samples refer to samples that are computed from directly measured variables. For instance, human faces are samples of a latent space of people photos. The inverse mapping of generated data to latent space samples helps with learning important features in latent space samples. To learn such features, BiGAN includes an encoder besides the generator and discriminator. In addition to distinguishing between real samples and synthetic ones, the discriminator in BiGAN is tasked to distinguish between encoding created by the encoder and the encoding from the latent space created by the generator, i.e., if $G(E(x)) = x$ is true, where E is the encoder [60]. This is helpful to learn complex distributions of data. For example, given a latent distribution of photos, the encoder learns the most important features to reconstruct those photos or to use in classification models. The BiGAN objective is a minimax game as [60]:

$$\min_{G,E} \max_{D} V(D, E, G), \tag{1.15}$$

where

$$\begin{aligned} V(D, E, G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x, E(x))] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z), z))] \\ \mathbb{E}_{x\sim p_r(x)}[\mathbb{E}_{z\sim p_E(.|x)} \log D(x, z)] + \mathbb{E}_{z\sim p_z(z)}[\mathbb{E}_{x\sim p_G(.|z)} \log(1 - D(x, z))]. \end{aligned} \tag{1.16}$$

1.2.12 Forecasting Models

Linear Regression Linear regression is a statistical method for finding the relationship between a dependent variable and independent variables or features [48]. Let us consider a multiple linear regression with x_j independent variables [48]. The

estimated value $\hat{y}$ in the linear regression model follows

$$\hat{y} = \beta_0 + \sum_j \beta_j x_j + \varepsilon, \tag{1.17}$$

where β_j represents the model weights or regression coefficients and ε is the error. L1 regularization is the absolute sum of the model weights multiplied by a shrinkage value (λ). It is formulated as $\lambda \sum_j |\beta_j|$ and the regression model goal is to minimize

$$\sum_i \left(y_i - \beta_0 - \sum_j \beta_j x_{ij}\right)^2 + \lambda \sum_j |\beta_j|, \tag{1.18}$$

where y_i is the actual data point and x_{ij} is the value of the independent variables for each data point.

ARIMA The ARIMA model is used to forecast the time-series data [49, 50]. ARIMA consists of three models: auto-regressive (AR), integration, and moving average (MA). It uses three hyper parameters: p, d, and q, where p is the order of auto-regressive model, d is the degree of difference, and q is the order of moving average [49]. The ARIMA(p, d, q) is defined as

$$\left(1 - \sum_{k=1}^{p} \alpha_k L^k\right) (1 - L)^d X_t = \left(1 + \sum_{k=1}^{q} \beta_k L^k\right) \varepsilon_t \tag{1.19}$$

where α is the coefficient of discrete time linear equation of AR, L is a time lag operator defined as $LX_t = X_{t-1}$, X_t is the observed value at time t, β is the coefficient for the noise term, ε, in MA [51].

Long Short-term Memory In an MLP layer, the inputs and outputs are not interconnected with each other. Unlike MLPs, the recurrent neural networks (RNNs) [61] learn the time evolution of data sequences by having outputs dependent on inputs. This dependency creates a memory that includes the calculations already made by the RNN from data in a previous timestep. The main disadvantage of RNNs is that they only learn short-term dependencies of data. To overcome this, Long Short-term Memory (LSTM) models were introduced [61]. LSTM has memory cells that include three modules called input gate, output gate, and forget gate. The calculations from previous timesteps are remembered in these memory cells. A gate is a MLP layer with a sigmoid activation function, $\frac{1}{1+exp(-x)}$. The input gate decides the new information to be added to the memory cell. The forget gate decides what information to be omitted from the memory cell. The output gate decides on the memory-cell information to output. Unlike RNNs, the memory cell in LSTM keeps long-term dependencies as well as short-term ones. Therefore, LSTMs are capable of learning more complex time evolution of data. LSTMs are more suitable than RNNs to predict time-series data. An example of time-series data is the number of subway entries per day and their predictability of the number of COVID-19 cases, as discussed in [50].

1.2.13 Kolmogorov-Smirnov Two Sample Test

The Kolmogorov-Smirnov two-sample test measures if the commutative distribution function (CDF) of two sets of data have the same distribution [62]. Let us assume that the observed CDF of one sample of size m is $S_m(X) = \frac{k}{m}$, where k is the number of data points less than or equal to X [63]. The other sample CDF is $S_n(X) = \frac{k}{n}$ with a size of n. Then, the Kolmogorov-Smirnov two-sample is defined as

$$D_{m,n} = \max |S_m(X) - S_n(X)| . \tag{1.20}$$

The null hypothesis states that the two distributions are the same. This hypothesis is rejected at a level α, if $D_{m,n}$ is larger than $c(\alpha)\sqrt{\frac{n+m}{nm}}$, where $c(\alpha) = \sqrt{-\frac{1}{2}\ln(\frac{\alpha}{2})}$. For instance, for $\alpha = 0.05$, the $C(\alpha) = 1.358$.

1.2.14 Performance Metrics

To evaluate the performance of ML algorithms, we use precision, recall, and F1 score. Precision represents the number of correctly classified items of the positive group (e.g., application A) among the total number of data points classified as positive. The precision of a classification problem is defined as $\frac{TP}{TP+FP}$, where True positives (TP) are the correctly classified traffic of application A as belonging to A and false positives (FP) are the incorrectly classified traffic of application B as belonging to A. Recall represents the number of correctly classified items among the relevant data points. Relevant data points are the union FN and TP data points. False negatives (FN) are the incorrectly classified traffic of application A as belonging to B. True negatives (TN) are the correctly classified traffic of application B as belonging to B. Recall is defined as $\frac{TP}{TP+FN}$. F1-score is the harmonic mean of precision and recall for each group of data. The F1-score is defined as $2\frac{(\text{Precision})(\text{Recall})}{\text{Precision+Recall}}$. To calculate an overall average precision, recall, and F1-scores of different tests, we calculate the micro and macro averages [64] of each metric. Considering a dataset consisting of n different classes (e.g., traffic of n different applications), we calculate the micro average precision of all classes as

$$\text{Micro average precision} = \frac{\sum_{j=1}^{n} TP_j}{\sum_{j=1}^{n} TP_j + \sum_{j=1}^{n} FP_j}, \tag{1.21}$$

where TP_j and, FP_j are the TP and FP of each class, respectively. Similarly, we calculate the micro average of recall as

$$\text{Micro average recall} = \frac{\sum_{i=1}^{n} TP_j}{\sum_{j=1}^{n} TP_j + \sum_{i=1}^{n} FN_j}, \tag{1.22}$$

where FN_j is the FN of each class. Micro average of F1-score is the F1-score of micro averages of precision and recall. In multi-class datasets, comparing micro averages with per-class statistics help demonstrate the overall performance of the system [64].

The macro average precision of all classes is defined as

$$\text{Macro average precision} = \frac{\sum_{j=1}^{n} \text{Precision}_j}{n}. \tag{1.23}$$

Similarly, we calculate the macro average of recall as

$$\text{Macro average recall} = \frac{\sum_{j=1}^{n} \text{Recall}_j}{n}. \tag{1.24}$$

1.3 Countering Internet Traffic Classifiers

In this dissertation, we study the countermeasures against adversarial ITCs. The majority of ITCs focus on traffic classes such as web browsing, emails, P2P, streaming, gaming, and file transfer. In this work, we specifically target ITCs that aim to identify users' software such as Google Chrome and WhatsApp. We aim to protect users' privacy against adversarial ITCs designed to detect the applications generating network traffic. We propose different traffic modification methods to secure it against ITCs. These methods are traffic statistics equalization and GAN tunnel.

1.4 Traffic Statistics Equalization

We propose using three traffic modification methods: equalized packet length, equalized packet count, and equalized inter-arrival times in TCP flows. A flow is the set of packets generated by an application of a computing system that is identifiable by source/destination IP addresses and port numbers [2]. We study the effects of these equalization methods against ITCs based on different ML algorithms to determine the best countering method against each algorithm.

The equalized packet count method is the most effective one against most studied ML algorithms in adversarial ITCs. This equalization method decreases the average precision and recall of tree-based classifiers such as Random Forest and XGBoost by about 25%. The tree-based classifiers are most effective in identifying user applications as they are capable of capturing slight differences in network traffic among different applications [1]. Against other adversarial classifiers such as SVM, the equalized packet counts decrease the average precision and recall of these classifiers up to 82%. The equalized packet length is the second effective method to counter adversarial ITCs and capable of decreasing the average precision of ML algorithms from 15% and up to 81%. The equalized inter-arrival times method is the least effective method among the proposed equalization methods and can decrease the precision of ML algorithms between 0% to 21%.

1.5 GAN Tunnel

The proposed countering methods against ITCs are not capable of masking the network traffic entirely against various ML algorithms, as each ML algorithm may favor a different traffic feature for classification [2]. Therefore, to design a method that does not rely on specific traffic masking features, we propose using a GAN model [53]. GANs are originally used to generate fake images that are indistinguishable from real ones. In this study, we design a GAN model that generates traffic flows, similar to

real ones generated by six different applications. We use the generated flows to mask the actual network traffic using a tunnel that we call the GAN tunnel.

The GAN tunnel encapsulates the real network traffic into packets of generated flows representing a different application, i.e., a decoy, to avoid detection by ITCs. To test the efficiency of the GAN tunnel, we use Random Forest [42] and XGBoost [65] to detect the source application of transmitted data. We show that the GAN tunnel traffic is entirely identified as the intended decoy application instead of the source application.

1.6 Summary of Contributions

Our main contributions in this research are: 1) comparing state-of-art ML classifiers accuracy for Internet traffic classification, 2) proposing different methods to mask the traffic and secure user privacy online, 3) comparing ML classifiers accuracy for the proposed masking methods to find the most suitable method, 4) evaluating Internet classification and masking accuracy on newly generated traffic, 5) designing a network traffic generator based on GAN, 6) propose a novel method for traffic tunneling, and 7) compare the performance of different ITCs against the GAN tunnel traffic.

CHAPTER 2

TRACKING USER APPLICATION ACTIVITY BY USING MACHINE LEARNING TECHNIQUES ON NETWORK TRAFFIC

A network eavesdropper may invade an online user's privacy by collecting the passing traffic and classifying the applications that generated the network traffic. This collection may be used to build fingerprints of the user's Internet usage. This chapter investigates the feasibility of performing such a breach of encrypted network traffic generated by actual users. We adopt the Random Forest algorithm to identify Internet applications in use by a campus network user. Our classification system identifies and quantifies different statistical features of a user's network traffic rather than looking into packet contents. The application profiling is performed even on encrypted traffic. Also, application classification is performed without employing a port mapping at the transport layer. Our results show that an application can be identified with an average precision and recall of up to 99%.

A wide variety of machine learning algorithms have been studied for Internet traffic classification [5, 12]. These algorithms include: Naive Bayes [3, 4, 22], Bayesian classifier [3, 4, 16], Random Forest [3, 18], decision tree [3, 4, 21, 24], Naive Bayes tree [3, 4], MLP [3], SVM [6, 8, 9, 19, 20], and neural networks [7, 9]. Out of all these methods, Random Forest [42] provides the highest accuracy in our tests. Therefore, we adopt this algorithm in this chapter.

To classify applications, we use the header fields of the TCP and Internet Protocol (IP), including the source and destination port numbers and the IP addresses of the captured packets. Our machine-learning-based classification uses statistical properties of the generated traffic rather than the content carried by the traffic. The Random Forest algorithm identifies the statistical properties of the traffic, such as the length, inter-arrival time, and the TCP window size of these packets to build a profile for each application and user. We aim to classify applications in the following

categories: web browsing, cloud storage, messaging, note-taking, and data streaming. Based on our captured data, the captured packets for applications in these categories differ in the count, average size, inter-arrival times, and other statistical features. For instance, average packet sizes in cloud storage and streaming categories are larger than the messaging and note-taking categories.

2.1 Online Activity Tracking

The Random Forest classifier consists of a stack of tree predictors. We call each tree predictor a *tree* in the remainder of the chapter, for simplification. For each tree, a random vector Θ_i is generated, where i denotes the index of the tree. Θ_i is generated independently of previously generated Θ_is. By using a training dataset and Θ_i, makes ith tree grow and be able to classify different classes of data or applications. Each tree leaf is a class of input vectors. The nodes of a tree test the attributes of each class with a branch for each possible outcome [66]. The ith tree is a classifier denoted as $h(\mathbf{x}, \Theta_i)$ where $\mathbf{x}$ is an input vector or a data point to be classified. In this chapter, the input vectors comprise of the statistics features of flows of traffic [66]. Each tree chooses the most popular class (label) of $\mathbf{x}$ as an output. Therefore, a Random Forest classifier is also defined as a collection of trees $h(\mathbf{x}, \Theta_i)$ with independent identically distributed random Θ_is, where the statistical mode of trees outputs is selected as the classified class of input $\mathbf{x}$ [42].

In our approach, we first extract a feature set from the header fields of packets in TCP flows. A TCP flow is defined as a stream of TCP packets exchanged between two end nodes (e.g., a user application and its server) having the same source and destination IP addresses, port numbers in each forwarding direction. Our feature set includes packet length, packet inter-arrival time, TCP window size, port numbers, and source and destination IP addresses. The Random Forest classifier is trained using these features of the captured TCP flows.

In this chapter, we use Microsoft Windows event-tracing service [38] for labeling the application-specific traffic. By resorting to labeling, we enable the use of supervised machine learning techniques for traffic classification. The trained model is then used to classify the user applications by using newly generated traffic.

Figure 2.1 shows an example of Internet traffic classification using a Random Forest classifier. Here, the captured TCP flows, with their extracted sets of features, are fed to a classifier to identify Google Chrome and WhatsApp applications. This classifier uses different trained decision trees to classify the traffic, and the final result is calculated as the mode of the results from all decision trees. In this example, most of the trees classify the traffic as generated by Google Chrome web browser, and therefore, the final decision or the statistical mode of the output of trees is the Google Chrome web browser.

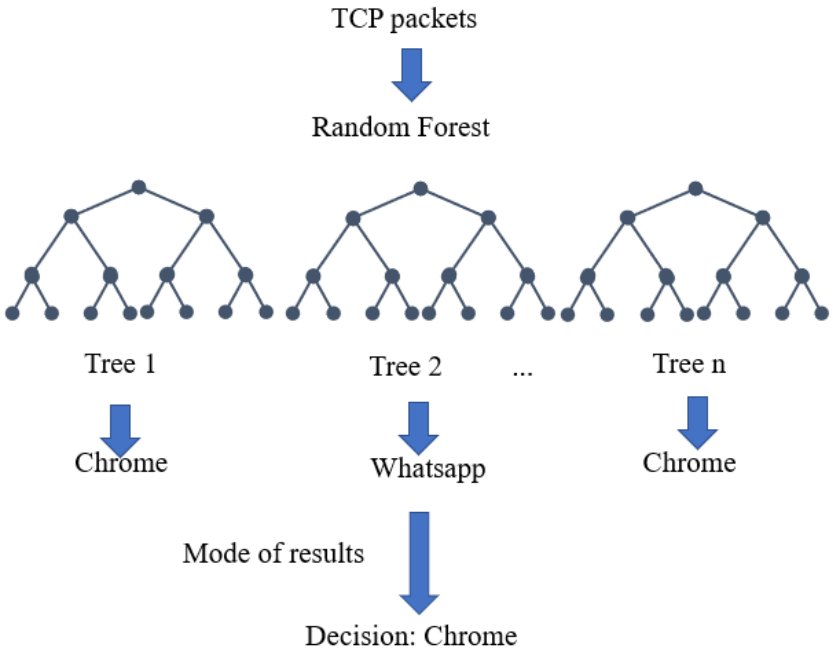

Figure 2.1 Random Forest for traffic classification.

Figure 2.2 shows an example of a scenario on how machine learning may be used for application identification through traffic classification by an adversarial network

user. Here, a network user is connected to the Internet, and an attacker captures a copy of transmitted packets for classification. The attacker uses a trained machine-learning algorithm to classify the copied traffic to identify the applications used by the legitimate user and possibly track the online user activity.

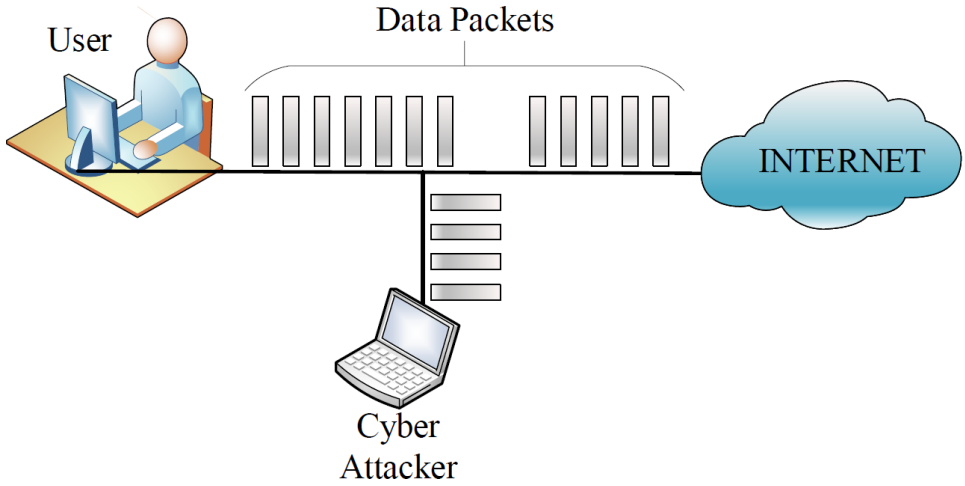

Figure 2.2 Representation of application identification made by an attacker external to a user's network.

We use actual network traffic for this study, where 80% of the data is used for training and the rest for testing. Our dataset includes traffic traces captured from six different users in a (university) campus network. We use a modified version of Wireshark packet sniffer [36] called Wireshark PAINT [37], which uses event tracing for Microsoft Windows (ETW) [38], to capture labeled traffic. ETW is capable of associating the generated network traffic with the issuing process, and therefore, identifying the originating application. We captured data for 28 days from these different users. The total generated dataset consists of about 11 million TCP packets.

Figure 2.3 shows a detailed overview of our data process. The three main phases in our system are: preprocessing of data, training of the classifier, and actual classification. The TCP packet headers are extracted from the captured traffic and organized for the classifier in Step 1. These traces of header data are grouped to form

TCP flows in Step 2. The extracted data is then used to calculate the mean, median, and variance of the following parameters for each flow in Step 3:

1. Packet inter-arrival time
2. Packet length
3. Window size

and the following statistics:

1. Number of packets (received and sent)
2. Source/Destination IP addresses
3. Source/Destination port numbers

In this step, flows are also labeled to train our Random Forest algorithm. Step 4 shows the training dataset and some of the included extracted features. In Step 5, our machine learning model is trained using the Scikit-Learn [40] library, in python. In Step 6, or the classification phase, the captured data of a user is used as the input (Step 7) for classification. In Step 8, the classified data of the user, as the result of our classifier, is obtained.

We use precision, recall, and F1-score as the evaluation metrics. Figure 2.4 shows a graphical representation of true positive, true negative, false positive, and false negative values that are used to calculate our evaluation metrics. In this figure, the circular set in the middle contains the data points that are positively classified (i.e., true and false positives). All the data points outside the circular set in the same figure are negatively classified (i.e., true and false negatives). The data points that are in the top portion of the circular set, which is colored in red, are falsely classified as positive (i.e., false positives). The data points outside the circular set and inside the lower part of the figure, which are colored in light green, are falsely classified as negative (i.e., false negatives). For example, let us assume a scenario where a user

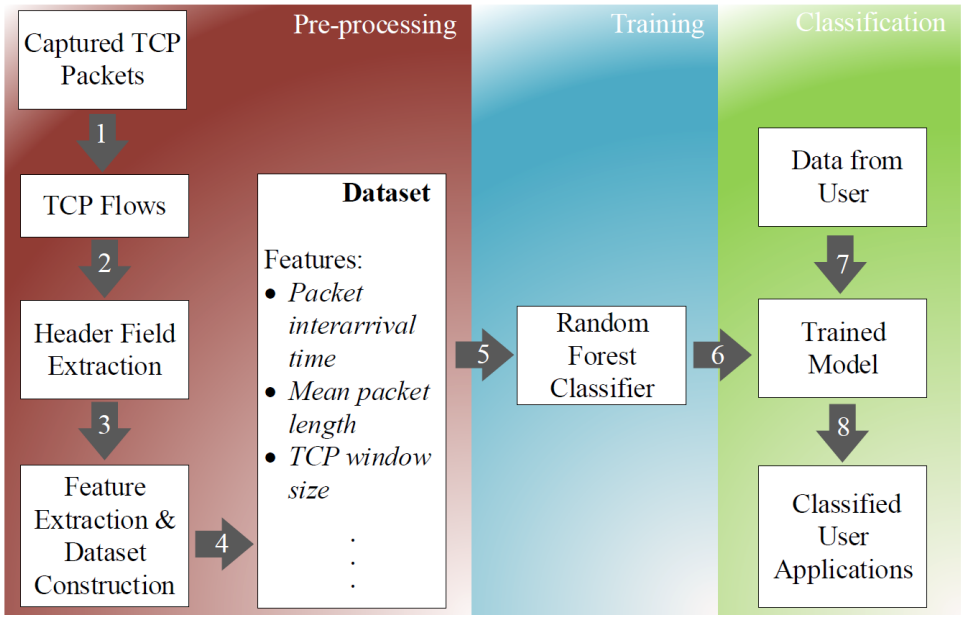

Figure 2.3 Detailed classification process.

uses two applications; applications A and B, and we aim to classify all data packets generated by application A. True positives (TP) are the correctly classified traffic of application A as belonging to A. False negatives (FN) are the incorrectly classified traffic of application A as belonging to B. True negatives (TN) are the correctly classified traffic of application B as belonging to B. And finally, false positives (FP) are the incorrectly classified traffic of application B as belonging to A. Precision of a classification problem is defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.1}$$

Precision represents the number of correctly classified items of a group (e.g., application A) among the total number of data points classified as positive. Recall represents the number of correctly classified items among the relevant data points as

Figure 2.4 shows. Recall is defined as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.2}$$

Relevant data points are the union FN and TP data points. F1-score is the harmonic mean of precision and recall for each group of data. The F1-score is defined as:

$$\text{F1-score} = 2\frac{(\text{Precision})(\text{Recall})}{\text{Precision} + \text{Recall}} \tag{2.3}$$

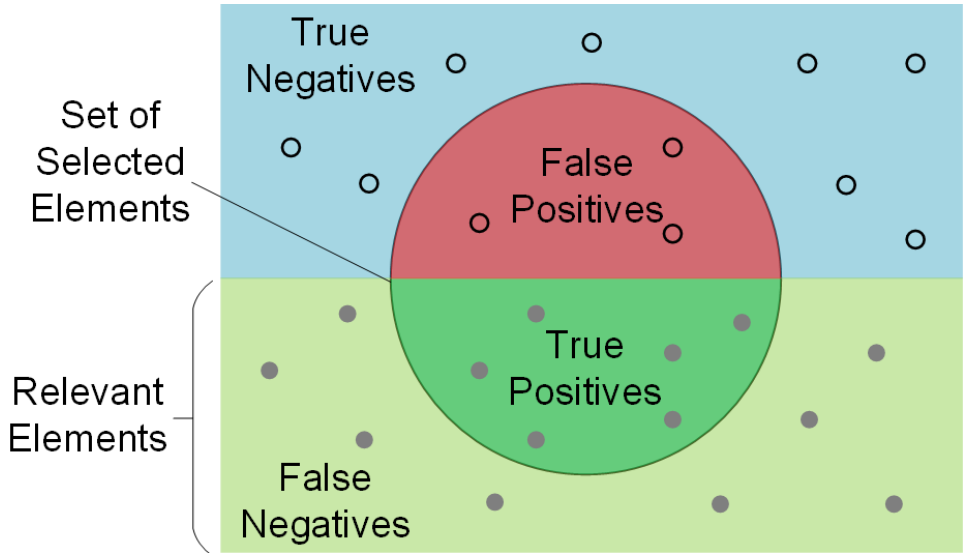

Figure 2.4 Graphical representation of true positives, false negatives, true negatives, and false positives.

2.2 Classification Evaluation

We used 29,617 TCP flows captured from different applications for our evaluations. The applications considered in this chapter and their categories are listed in Table 2.1. These applications are selected because of their popularity [67, 68].

Table 2.1 Applications and their Categories

Category	**Applications**
Web browsing	Google Chrome
Cloud storage	Google Drive and One Drive
Messaging	WhatsApp
Note taking	Microsoft One Note
Streaming	Spotify

As an evaluation scenario, we split the whole captured dataset into two smaller datasets, one for training and the other for testing. The training dataset is about 90% of all the captured data, and the remaining is used as test data set. We call this evaluation *split-train-test dataset evaluation*. The top 10 most distinguishing features of the captured packets of each flow are the following:

1. Variance of packet inter-arrival times
2. Maximum of packet inter-arrival times
3. Mean of packet inter-arrival times
4. Median of window size
5. Mean of window size
6. Variance of window size
7. Mean of packet length
8. Flow packet count
9. Sum of packet lengths in a flow
10. Median of packets length

Out of the distinguishing feature list, the packet inter-arrival times, TCP window sizes, packet lengths, and packet counts are the most dominant identifiers.

Figure 2.5 shows the precision, recall, and F1-score of our classifier for each application. Our classification model achieves an overall average precision of 91%, with the lowest and highest precisions of 82 and 97%, respectively, among the applications used for classification, as summarized in Table 2.2. The Chrome web browser has the highest precision and recalls among all applications. Therefore, 96% of the packets classified as Chrome web browser are actually generated by the Google Chrome web browser. Additionally, 10% of packets generated by other applications are falsely classified as Chrome web browser's traffic. Google Drive desktop client is classified with a precision of 82%; which is the lowest obtained precision among all applications considered in the evaluations.

Table 2.2 Results of the Split-train-test Dataset Evaluation

Application	**Precision**	**Recall**	**F1-score**
Chrome	0.90	0.96	0.93
Google Drive	0.82	0.69	0.75
One Drive	0.84	0.65	0.73
One Note	0.92	0.96	0.94
Spotify	0.97	0.87	0.91
WhatsApp	0.96	0.83	0.89
Micro Average	0.91	0.91	0.91

Figure 2.6 shows the normalized confusion matrix of the classified applications. The confusion matrix shows the TP and FP rates per application. Here, Google Chrome browser, having generated the largest number flows, has a TP of 96%. About 2% of Google Chrome packets were misclassified as Google Drive traffic. The Google Drive application has a TP of 69%. Moreover, 27% of its packets were misclassified as Google Chrome browser packets. Microsoft One Drive application has a TP of 65%. Also, 11 and 16% of the packets generated by Microsoft One Drive were misclassified as Microsoft OneNote and Google Chrome, respectively.

To demonstrate the system detection ability for newly generated traffic, the trained Random Forest classifier is used to identify the user applications by

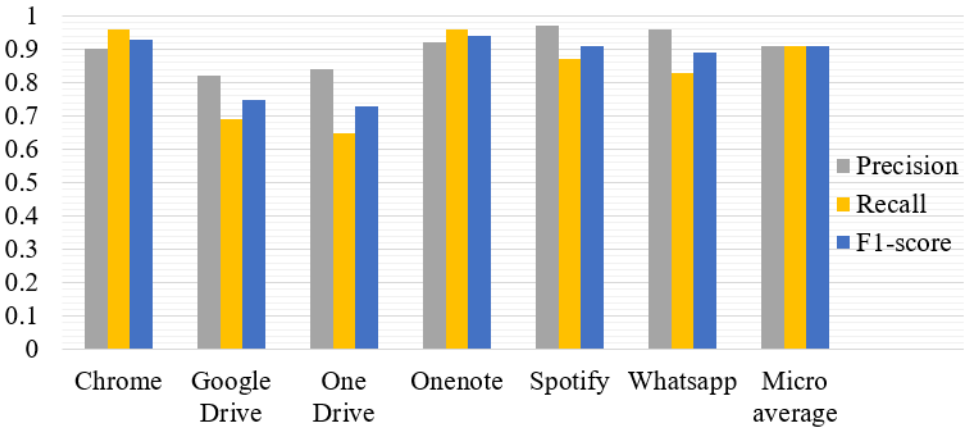

Figure 2.5 Precision, recall, and F1-score in our classifier for each application in the split-train-test dataset evaluation.

considering the packets not included in the training dataset. We call this evaluation *unknown traffic evaluation*. The test data includes one hour of captured traffic generated by applications used in the training phase. Figures 2.7 and 2.8 show the classification results of this dataset.

In this evaluation, our classifier achieves an average precision, recall, and F1-score of 95% each for all applications considered in this chapter. Individually, WhatsApp traffic has the highest precision, or 99%, among all other applications with a recall of 95%. Spotify has the lowest precision, or 91%, among all the applications with a recall of 74%, as Figure 2.7 shows. Table 2.3 lists the precision, recall, and F1-score of this evaluation. The overall scores of the applications in this evaluation are slightly better than those observed in the split-train-test evaluation.

Figure 2.8 shows the confusion matrix of the results for unknown traffic evaluation. Compared to the split-train-test dataset evaluation, the overall TP rates of the applications in the unknown traffic evaluation are higher. Spotify traffic has the lowest TP of 74% with 18% of its traffic misclassified as Google Chrome's.

The proposed model was evaluated on the detection of traffic generated by Chrome and WhatsApp applications for a duration of one hour of captured traffic.

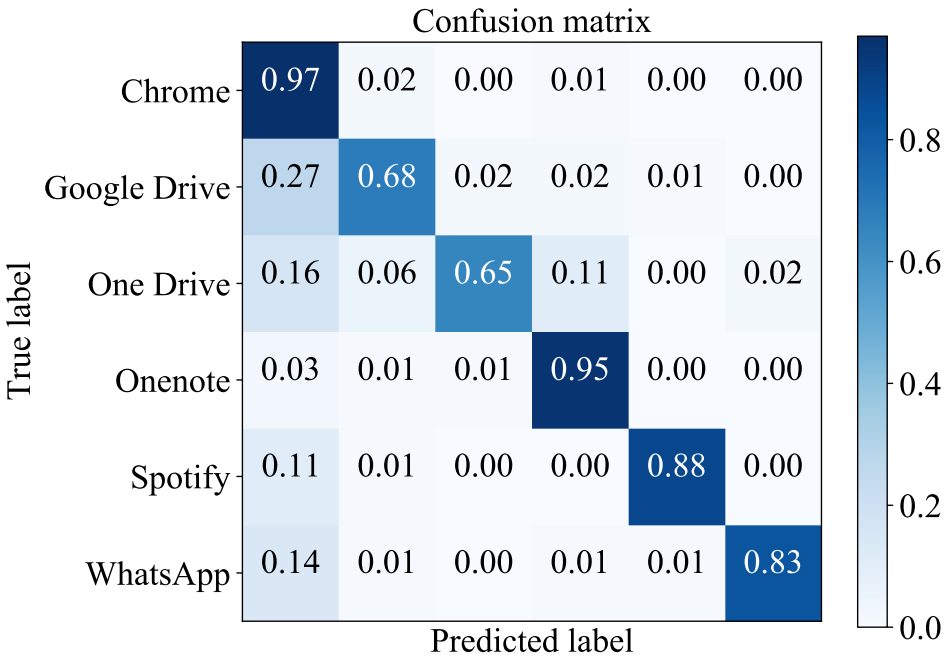

Figure 2.6 Normalized confusion matrix of the classified applications in the split-train-test dataset evaluation.

Here, we demonstrate that a small subset of applications can be classified with high accuracy. Figures 2.9 and 2.10 show the classification results of this dataset.

The Chrome web browser in this evaluation has a precision, recall, and F1-score of 100, 99, and 99%, respectively. The precision, recall, and F1-score of WhatsApp are 100, 96, and 98%, respectively. These results show that both WhatsApp and Google Chrome applications are correctly classified with a TP of 99 and 96%, respectively. Additionally, among the packets classified as either Google Chrome or WhatsApp, 99% of them belong to their respective groups. Therefore, the calculated micro average precision and recall of these applications are 99% each, as Figure 2.9 shows.

2.3 Conclusion

In this chapter, we proposed to use Random Forest classifier for Internet traffic classification and user application identification. We captured real traffic from six users in a campus network for about 28 days. Our model identifies user application

Table 2.3 Results of the Unknown Traffic Evaluation

Application	Precision	Recall	F1-score
Chrome	0.94	0.99	0.96
Google Drive	0.96	0.78	0.86
One Drive	0.96	0.79	0.87
One Note	0.97	0.96	0.97
Spotify	0.91	0.74	0.82
WhatsApp	0.99	0.95	0.97
Micro Average	0.95	0.95	0.95

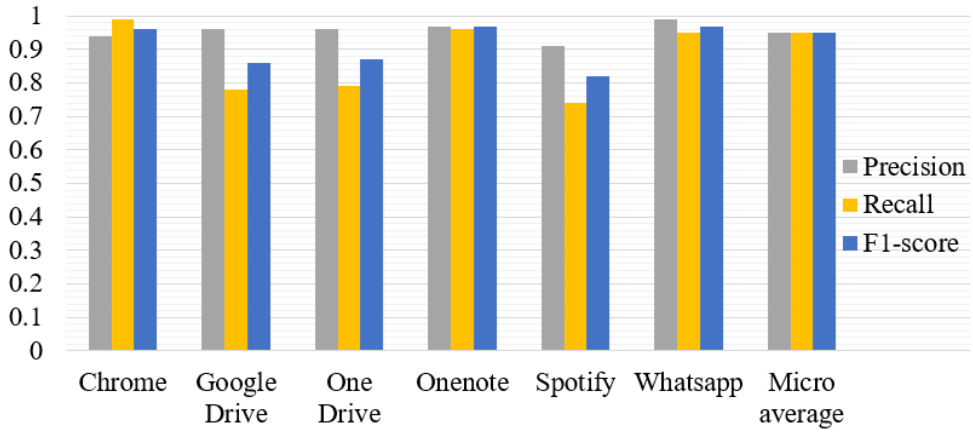

Figure 2.7 Precision, recall, and F1-score in our classifier for each application in the unknown traffic evaluation.

even from network packets with encrypted payloads. Our results show that the Random Forest classifier identifies the evaluated applications with precisions in the range of 82 to 94%. The packet inter-arrival time, TCP window size, packet length, and packet counts are identified as the most dominant identifiers for the classification of user applications. Therefore, obfuscating these statistics may help protect users' privacy. Our developed model automatically extracts the features of captured network traffic and trains the employed classifier using the collected data. We performed application classification on newly-generated traffic. Our classifier detects user applications with precisions in the range of 91 to 99% in the unknown-traffic evaluation. For online user activities limited to few applications, we test our system

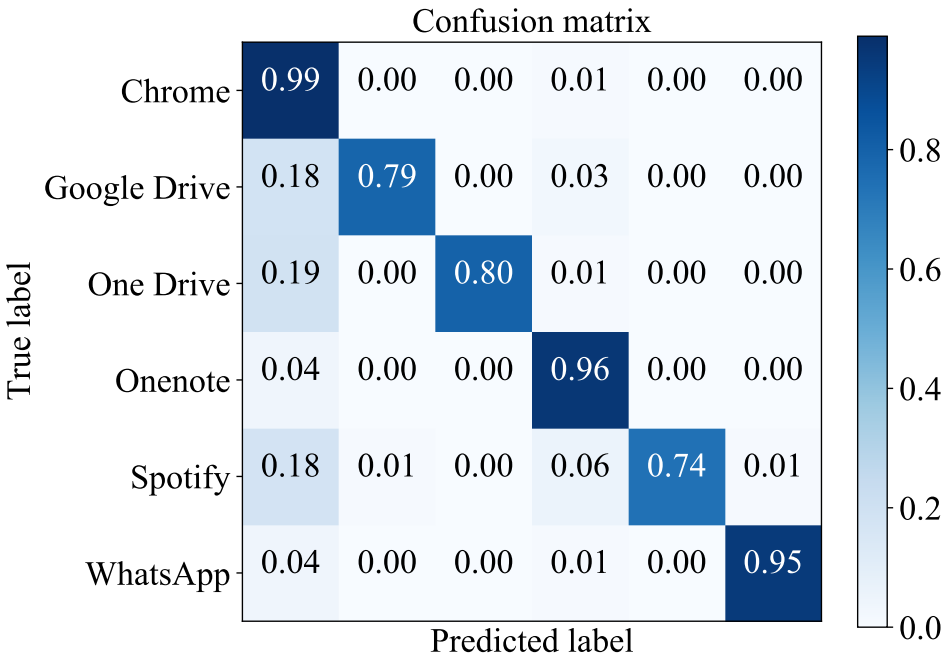

Figure 2.8 Normalized confusion matrix of the classified applications in the unknown traffic evaluation.

on a user dataset including two applications only, for which we achieve on average 99% precision and recall for each application in this evaluation.

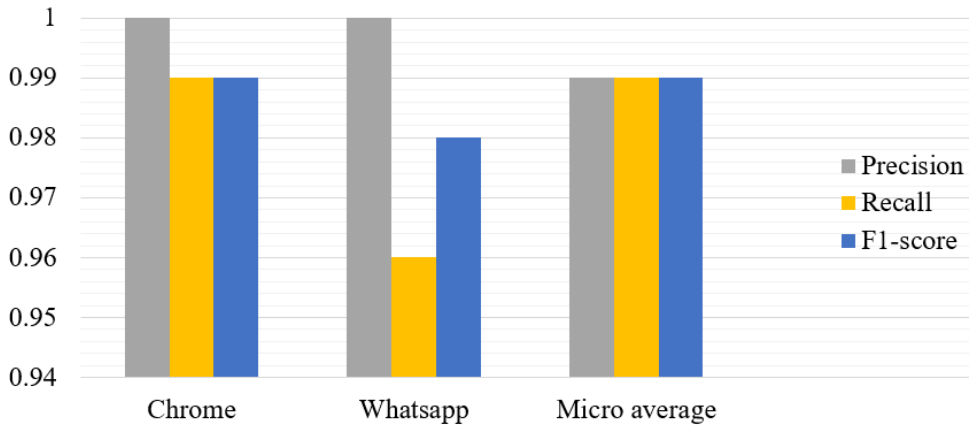

Figure 2.9 Precision, recall, and F1-score in our classifier for each application in the two-application traffic evaluation.

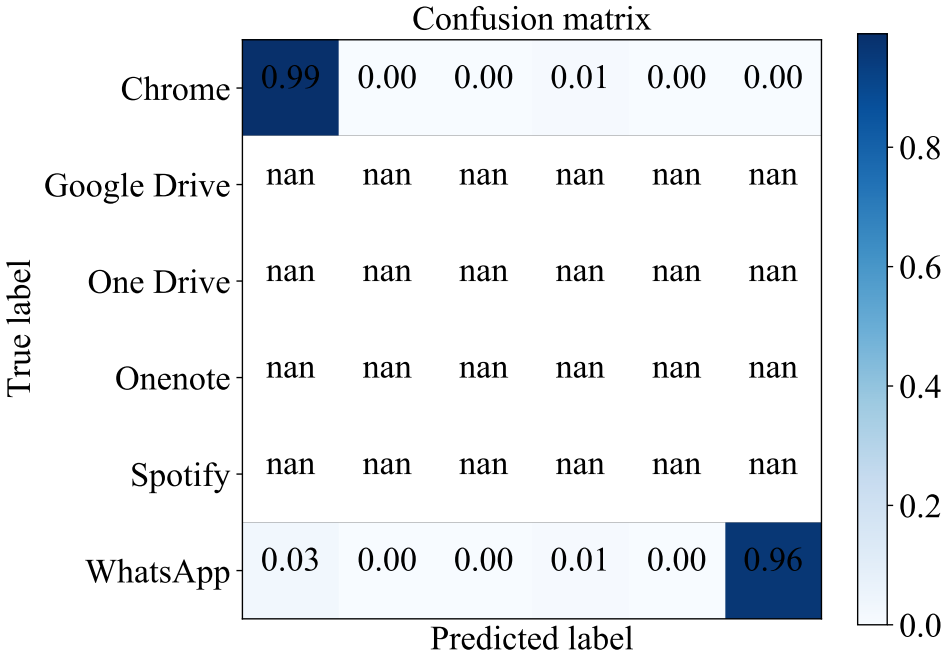

Figure 2.10 Normalized confusion matrix of the classified applications in the two-application traffic evaluation.

CHAPTER 3

COUNTERING INTERNET PACKET CLASSIFIERS TO IMPROVE USERS ONLINE PRIVACY

In this chapter, we introduce novel methods to block ITCs from detecting online user activities. We use state-of-art machine learning methods to detect the traffic patterns of exchanged traffic on the network for packet classification, and then we introduce our method to prevent classifiers from detecting user activities.

We focus on the statistical-based traffic classification method targeting user software usage online. The majority of ITCs focus on traffic classes such as web browsing, emails, P2P, streaming, gaming, and file transfer. We specifically target the classifiers which aim to identify the software used by users such as [18,26]. We use a combination of different traffic manipulation techniques to come up with a tailored solution for masking users' traffic and securing it against ITCs.

3.1 Background and Related works

3.1.1 Labeling Techniques

Labeling traffic traces refers to the association of an application generating a packet and the packet itself. Accessing users' computational resources is required to label the captured traffic. However, such access may be a cause of privacy concerns for the user [1].

Previous studies label traffic traces by matching packets' port numbers with the well-known port numbers listed by IANA, or port mapping. In these works, each packet is associated with a service such as HTTP, file-transfer protocol (FTP), Telnet, and domain name system (DNS) using the packet's port number [3, 4, 9, 16, 19, 22, 23, 71–74]. Many applications use the same well-known port numbers listed by IANA, such as port 80 used by HTTP or port 443 used by used by both SSL and VPNs. For instance, Google Drive and Microsoft OneNote packets use port 443 as the

destination port number to connect to their respective servers. To bypass firewalls of computational systems and networks, some applications such as Skype use the unrestricted port 443. As a result, port mapping limits traffic classification to groups of applications rather than software applications.

3.1.2 Countermeasures against ITCs

There are a number of works discussing countermeasures against Internet traffic classification [69, 75–78]. These works implement packet padding to counter ITCs. We provide a comparison of these works and ours in Section 3.4.2.

3.2 System Description

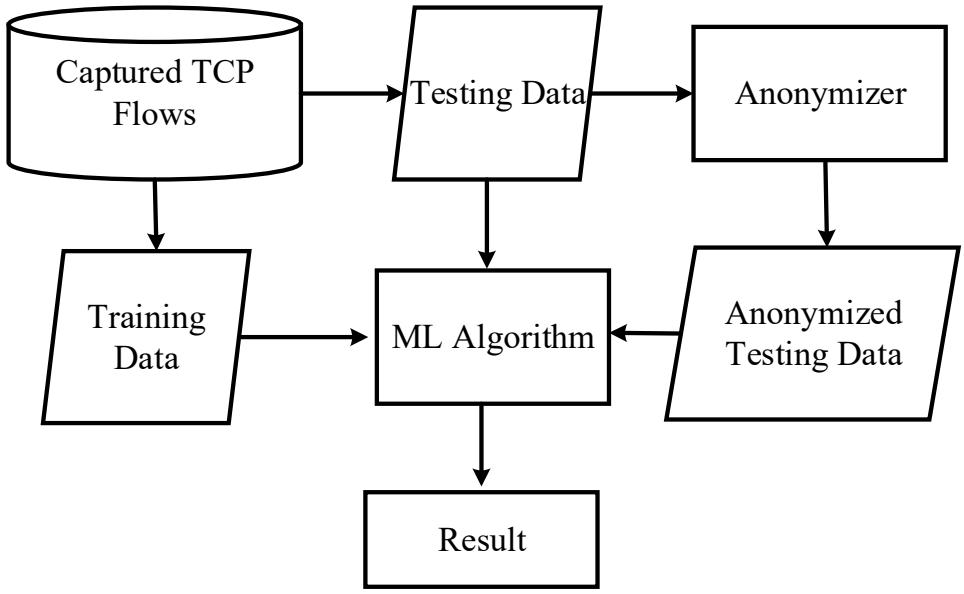

Figure 3.1 System model.

Figure 3.1 shows our system model. After capturing user data, our system extracts a list of pre-selected features from captured packets. We train our model with a 10-fold cross-validation in a stratified manner, in which the proportion of each class is kept the same for the whole dataset. In each cross-validation fold, the captured data is split into testing and training datasets with 10% for testing and 90% of data for training. The combination of over and under sampling balances training data classes

[79] using *Synthetic Minority Over-sampling Technique* (SMOTE) [80] combined with Tomek-links under-sampling method [81] implemented with *Imbalanced-learn* library [82]. The testing dataset is used to calculate the precision of each ML algorithm. A copy of the testing data is then used to create an anonymized testing dataset, which is used to determine how precision changes when the data are masked.

3.2.1 Dataset Details

Our dataset includes traffic traces captured from six different users in a university campus network for a collective duration of 28 days. The data were collected from April 2018 until December 2018 with an average size of 383 MB captured data for each day per user. We use a modified version of Wireshark packet sniffer [36], called Wireshark PAINT [37], which uses event tracing for Microsoft Windows (ETW) [38] to capture labeled traffic. ETW associates the generated network traffic with the issuing process, and therefore, it enables us to identify the originating application. To assign labels to each flow of traffic, we use a Python script to read the process names saved by Wireshark PAINT and associate them with each packet and flow when reading the pcap traces using the "dpkt" Python library [83]. We used Windows 10 computers with Intel Core i7 processors and laptops capable of running Wireshark PAINT to collect the data from ethernet links. We captured 8,038,520 TCP packets from the applications in Table 3.1. This table includes the information about number of flows and size of flows and packets. We extract a feature set from the header fields of packets in TCP flows. Here, we define a TCP flow as a stream of TCP packets exchanged between two end nodes that have the same source and destination IP addresses and port numbers in each forwarding direction. Our feature set includes the minimum, maximum, mean, median, and variance of the following parameters for each flow:

1. Packets inter-arrival times

2. Packet lengths

3. Packets TCP window sizes

and the following statistics:

1. Packet count

2. Total flow length

3. Source and destination IP addresses

4. Source and destination port numbers

3.2.2 Masking Methods

We aim to reduce the efficacy of adversarial ML classifiers used for identifying user's application software. Such classifiers analyze the traffic generated by the applications. To achieve this goal, we propose changing the statistics of the generated traffic generated by those applications. We focus on primary statistical features, such as packet size and, packet count of each TCP flow, and the inter-arrival times of the exchanged packets of a flow. We name these masking methods as equalized packet length, equalized packet count, and equalized inter-arrival times.

In the equalized packet length, we increase the size of each packet in test datasets to match the largest recorded packets in both directions of a flow. With the equalized packet count, we set the number of packets of each flow in test data to the maximum number of the captured packets in a flow. In equalized packet inter-arrival times, we space out the inter-arrival times of the packets transmitted in both directions to a size equal to the shortest inter-arrival time recorded in a flow. The traffic used in all our measurements is the same traffic with some statistics modified.

Table 3.1 Dataset Information.

Application	Number of flows	Total flow size (bytes)	Average flow size (bytes)	Average packet size (bytes)
Google Chrome	14,561	2,120,778,385	145,647.85	714.72
Google Drive	3,074	1,151,131,506	374,473.48	868.39
One Drive	756	22,780,297	30,132.66	328.67
OneNote	4,193	107,975,081	22,780,297	663.25
Spotify	3,179	3,218,263,686	1,012,350.95	957.36
WhatsApp	961	37,125,262	37,125,262	244.29

3.3 Evaluation Results

We evaluate the performance of our three equalization methods by applying them to two different test datasets. The first test dataset is split from training data. The second test data is the newly generated traffic. We call the newly generated traffic data as *unknown test data*.

We run our models on a PC with Microsoft Windows 10. This PC uses an Intel Core i7-8700K processor with 32 GB of memory and an NVIDIA GTX 1070 graphics card with 8 GB of dedicated memory. We use Scikit-learn library in Python [40] for ML classification.

To observe the differences between application features, we graph the empirical cumulative distribution functions (CDF) of lengths, counts, and inter-arrival times of packets. Figure 3.2 shows the empirical CDF of packet lengths of studied applications. To compare these graphs and CDFs, we use the Kolmogorov-Smirnov test [62] for each

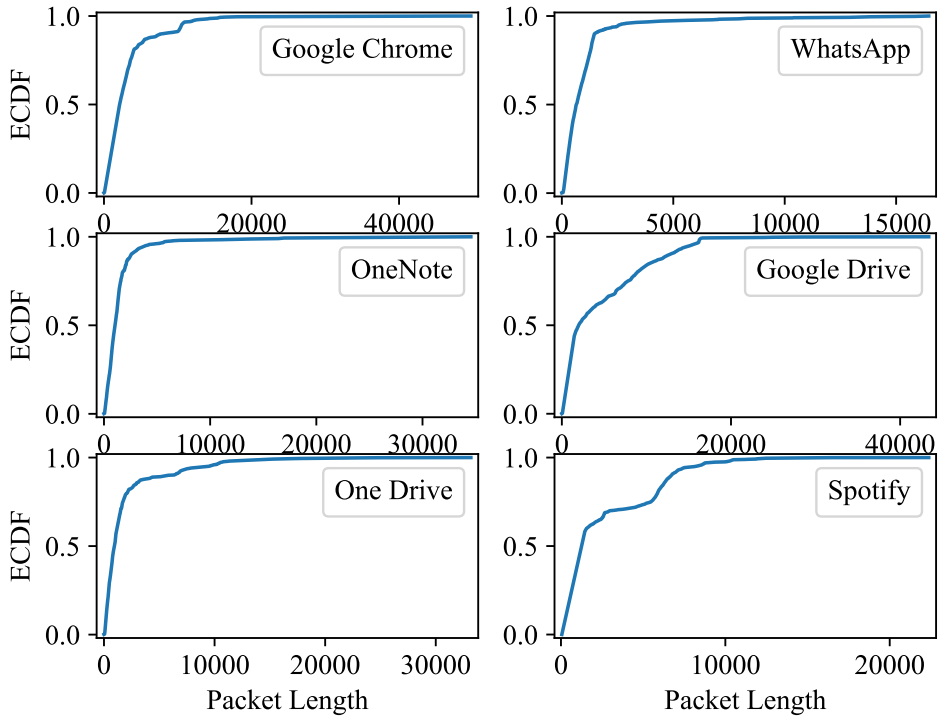

Figure 3.2 Empirical CDF lengths of packets generated by different applications.

pair of applications. In the two-tailed Kolmogorov-Smirnov test, the null hypothesis

is that the distributions of the two samples are the same. The calculated p-value for packet lengths of each application pair is 0, which is smaller than the significance level of 0.01. Therefore, we reject the null hypothesis. Based on this test, we conclude that packet length is a discriminating feature for distinguishing applications.

Figure 3.3 shows the empirical CDF graphs of packet counts for each application. To compare the plotted CDFs, we use the Kolmogorov-Smirnov test with the null hypothesis that the distribution of packet counts are the same for each application pair. Table 3.2 lists the calculated two-way p-values for this test. Since all the

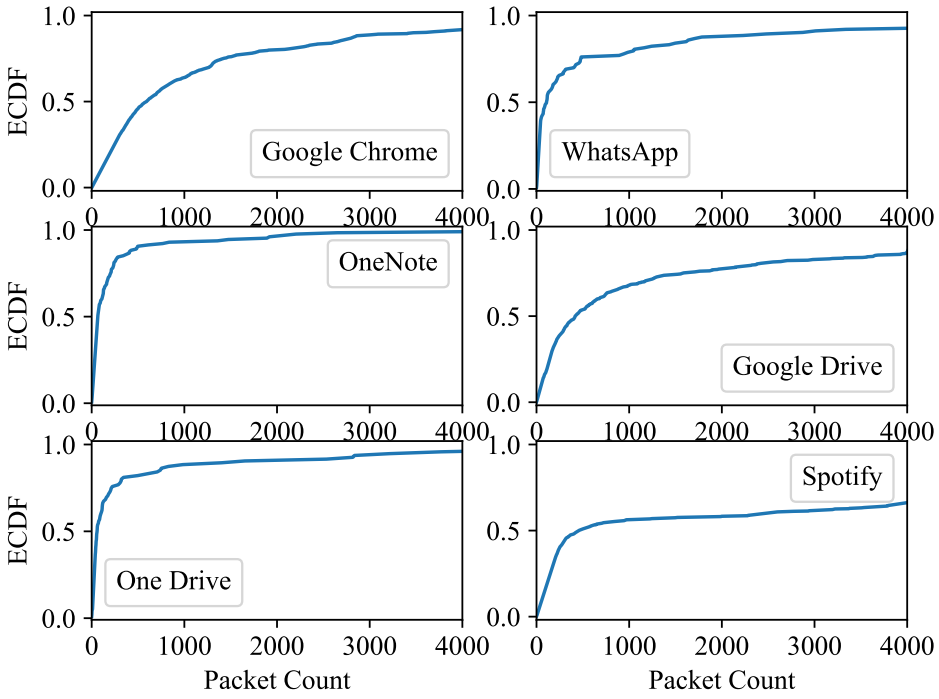

Figure 3.3 Empirical CDF of counts of packets generated by different applications.

p-values are smaller than 0.01, we reject the null hypothesis and conclude that the packet counts are also a discriminating classification feature for applications.

We compare the distribution of inter-arrival times of packets as Figure 3.4 shows. After using the Kolmogorov-Smirnov test, the calculated p-values for each pair of applications is 0 indicating that the distributions are indeed different for each application.

Table 3.2 Kolmogorov-Smirnov Test: Two-tailed P-values of Applications Packet Counts

	Google Chrome	**Google Drive**	**One Drive**
Google Chrome	1.0	3.643×10^{-209}	1.969×10^{-57}
Google Drive	3.643×10^{-209}	1.0	1.362×10^{-78}
One Drive	1.969×10^{-57}	1.362×10^{-78}	1.0
Onenote	0	1.496×10^{-289}	1.640×10^{-79}
Spotify	2.344×10^{-69}	1.670×10^{-41}	4.332×10^{-59}
WhatsApp	9.652×10^{-95}	7.285×10^{-138}	1.136×10^{-17}

	Onenote	**Spotify**	**WhatsApp**
Google Chrome	0	2.344×10^{-69}	9.652×10^{-95}
Google Drive	1.496×10^{-289}	1.670×10^{-41}	7.285×10^{-138}
One Drive	1.640×10^{-79}	4.332×10^{-59}	1.136×10^{-17}
Onenote	1.0	1.534×10^{-234}	2.395×10^{-204}
Spotify	1.534×10^{-234}	1.0	1.781×10^{-8}
WhatsApp	2.395×10^{-204}	1.7817×10^{-8}	1.0

3.3.1 Split Test Dataset

In this subsection, we evaluate our anonymization methods effect on classification of a split test dataset. Figure 3.5 shows the macro-averaged precision and recall of each ML algorithm on our dataset using stratified 10-fold cross validation sorted from the highest precision to the lowest one. The error bars show the 95% confidence interval of each result. Figure 3.5 shows that on the unmodified dataset, a number of tree-based algorithms, namely, Random Forest, XGBoost, and Voting achieve the best results followed by SGD and SVM. This result shows that these tree-based classifiers are suitable for multi-class classifications. Figure 3.9 shows the precision vs recall of each ML algorithm for each application before modifying the dataset. This figure shows the kernel density estimations (KDE) of precision and recalls for all the plotted data points as well. The KDE of both precision and recall are left-skewed, which shows that the majority of applications are classified with high precision and recall.

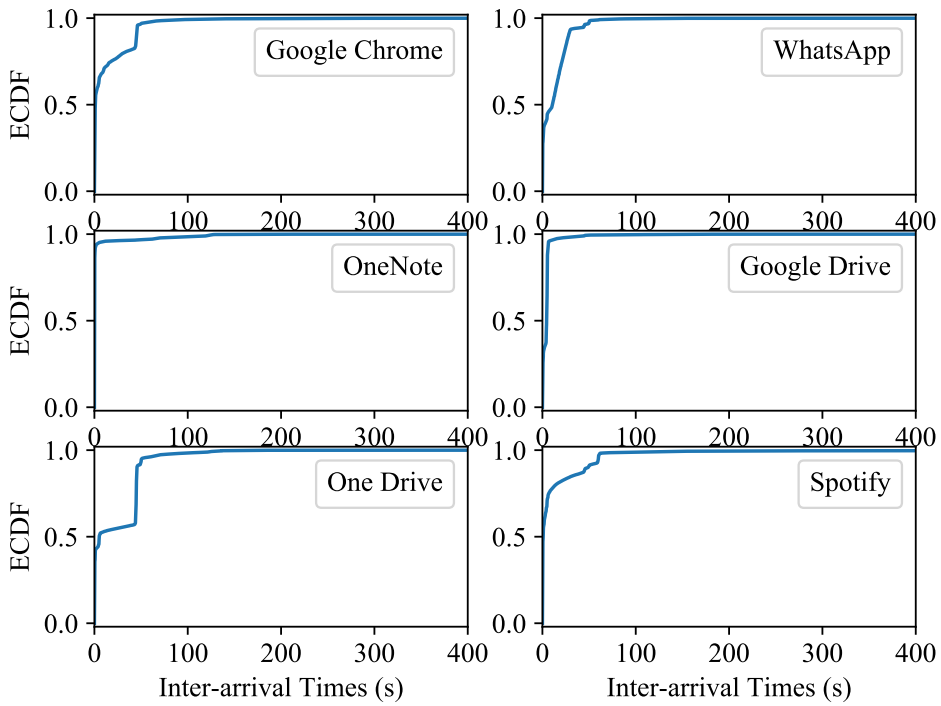

Figure 3.4 Empirical CDF of inter-arrival times of packets generated by different applications.

In the equalized packet length, we increase the packet lengths to the maximum TCP packet length transmitted by Google Chrome, or 65,535 bytes, to make the generated packets of other applications mimic web browsing traffic. Figure 3.6 shows the precision of each ML algorithm when packet length equalization is applied to the split test dataset. The top-performing classifiers on the original data, Random Forest, Voting, XGBoost, and SGD achieve precision between 0.68-0.75 before equalization. Among these four classifiers, XGBoost is affected the least from this equalization. SGD achieves a precision of 0.66±0.08 before equalization. However, it performs rather poorly on the modified test data with a precision of 0.05±0.03. The equalized packet lengths decrease the precision of ML algorithms.

Figure 3.10 shows the precision vs. recall of the ML algorithms for each application after applying equalized packet length. The KDE of precision is spread around 0.5 and is slightly right-skewed. Therefore, the ML algorithms classify the applications with a smaller precision on the modified traffic than on the original

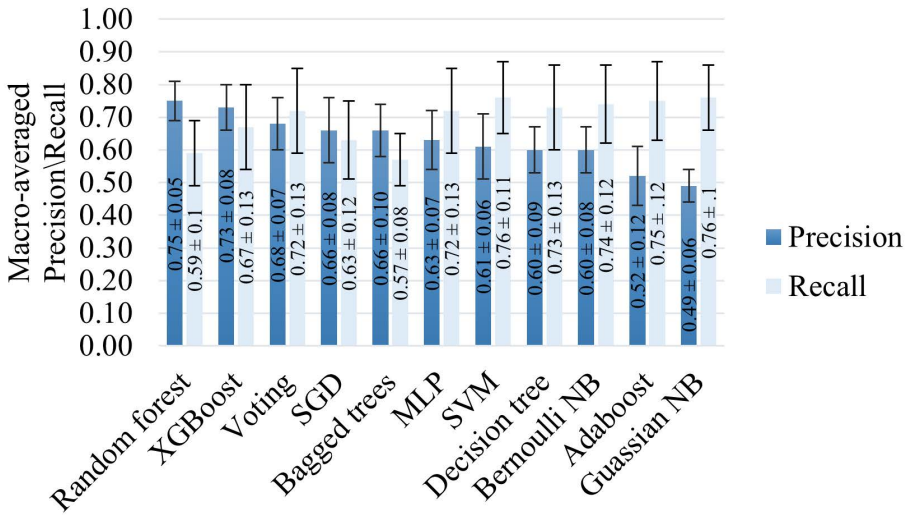

Figure 3.5 Macro-averaged precision of different ML algorithms on the split test dataset of original data

dataset. The KDE of recall here is right-skewed, meaning that most applications have a few true positives among the classified data. By comparing the KDE of precision and recall before and after padding, we conclude that this masking method can lower the overall performance of the ML algorithms.

Figure 3.7 shows the sorted macro-averaged results for the modified dataset with equalized packet count for each TCP flow. Compared to equalized packet length, this method is more effective for lowering the overall precision and recall of the ML algorithms. For instance, the precision of the Random Forest algorithm, which performs the best on the original dataset, drops from 0.75±0.05 to 0.50±0.07 after equalizing the packet count. The precision of XGBoost drops from 0.73±0.08 in the original data to 0.64±0.1. As Figure 3.7 shows, Bernoulli NB is the least affected ML algorithm in this modified dataset.

Figure 3.11 shows the KDE of precision and recall of ML algorithms for each application with equalized packet count. The distribution of precision and recall in this figure are right-skewed, showing that most ML algorithms achieve low precision and recalls. Here, WhatsApp, One Drive, and Google Drive applications,

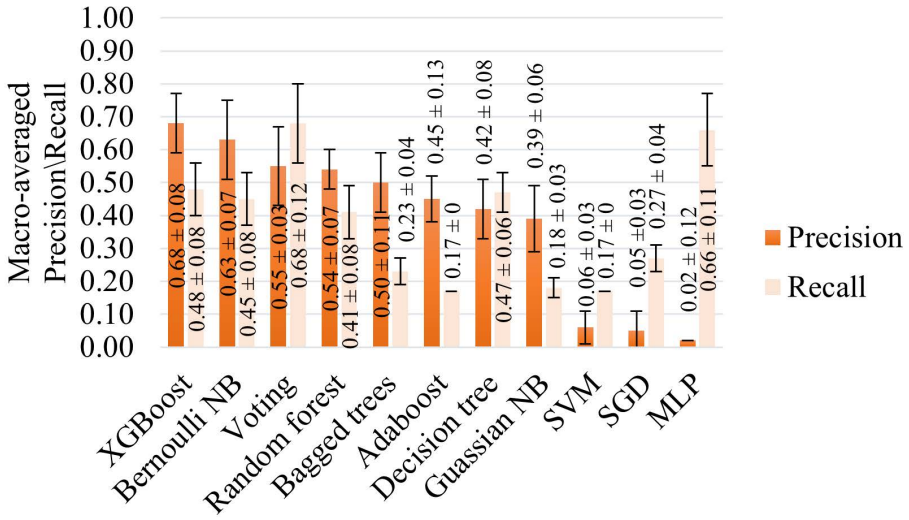

Figure 3.6 Macro-averaged precision of different ML algorithms on the split test dataset of equalized packet length

which are represented with gray, orange, and green markers, respectively, have precision and recall smaller than 0.5 in most cases, the lowest among all applications. Therefore, equalized packet count significantly lowers the detection probability of these applications. On the other hand, this masking method affect Google Chrome, OneNote, and Spotify.

We also tested equalized inter-arrival times on the split dataset. Figure 3.8 shows the macro-averaged precision and recall of ML algorithms for this test. By comparing Figures 3.5 and 3.8, we observe that equalized inter-arrival times is not as effective as the equalized packet length and packet count. The most affected ML algorithm here is XGBoost with a 0.21 drop in precision and recall as compared to the results on the original dataset. However, equalized inter-arrival times does not significantly affect the other ML algorithms, as these algorithms experience 0.5 or a smaller drop in their precision. Therefore, XGBoost relies on inter-arrival times of packets more than the other studied classifiers.

Figure 3.12 shows the KDE of precision and recall of the ML algorithms on the dataset with equalized packet inter-arrival times. The distributions of both precision

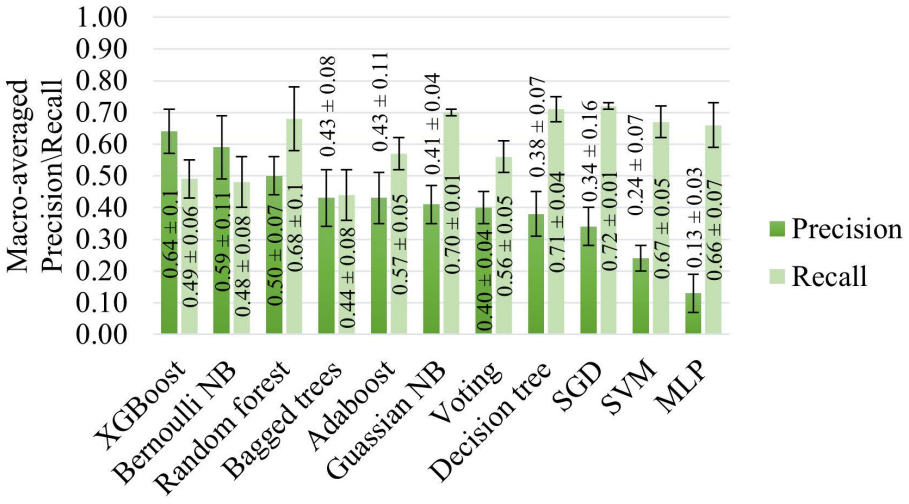

Figure 3.7 Macro-averaged precision of different ML algorithms on the split test dataset of equalized packet count

and recall are left-skewed. Therefore, this masking method is the least effective among the proposed three. This test affects the precision and recalls of One drive and Google drive more than other applications.

Table 3.3 F1-score Distributions Stats on the Split Dataset.

	Mean	Median	Mode	Skew
Original	0.617	0.655	0.75	-0.606
Equalized packet length	0.314	0.25	0	0.391
Equalized packet count	0.303	0.275	0.13	0.446
Equalized inter-arrival times	0.524	0.55	0.44	-0.43

Figure 3.13 shows the cumulative KDE graph of F1 scores of all studied applications and ML algorithms on split dataset, and Table 3.3 shows the calculated statistics of F1-score distributions. The cumulative KDE of the original data shows that the distributions of F1 scores are left-skewed, meaning that the studied ML algorithms perform well for classifying user software applications. Looking at the cumulative KDE reveals that equalized inter-arrival times slightly masks the user's activity from adversaries. Table 3.3 additionally shows that the distributions of

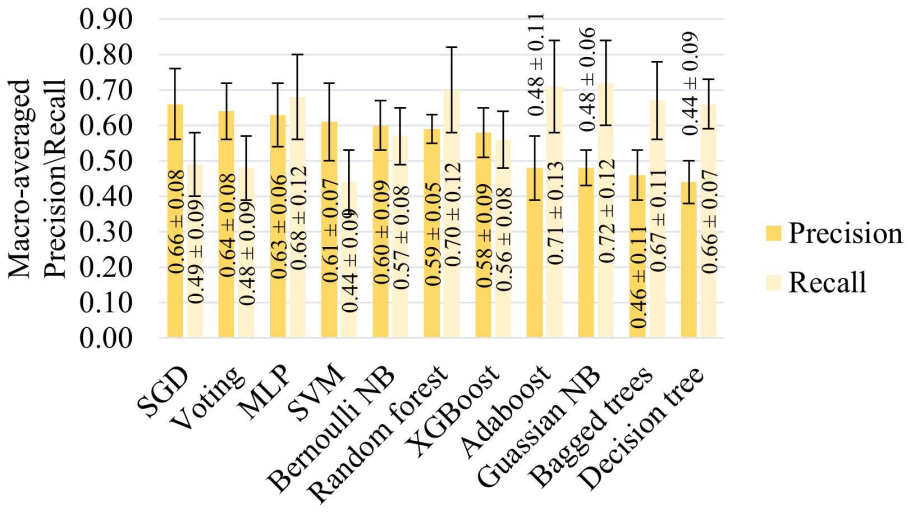

Figure 3.8 Macro-averaged precision of different ML algorithms on the split test dataset of equalized inter-arrival times

equalized packet length and equalized packet count are right-skewed, and therefore, they mask the user activity better than equalized inter-arrival times.

3.3.2 Unknown Test Dataset

In this subsection, we present the results of trained ML algorithms on the unknown test data. This dataset comprises about one-hour of captured traffic by a user using the six applications. Figure 3.14 shows the macro-averaged precision and recalls of ML algorithms on original test data. The results in this figure follow the ones from the split dataset as expected. The overall precision and recalls of ML algorithms on the unknown dataset are higher than the split dataset.

Figure 3.18 shows the classification results on the original dataset. Here, most ML algorithms achieve precision and recalls of over 0.79. XGBoost, Random forest, and Voting are the top three classifiers for the unknown dataset. Moreover, the distribution of precision and recalls of the ML algorithms are left-skewed, meaning that most applications have precision and recalls close to 1.0.

With equalized packet length, the performance of ML algorithms decreases. Figure 3.15 shows the macro-averaged precision and recalls of ML algorithms on the

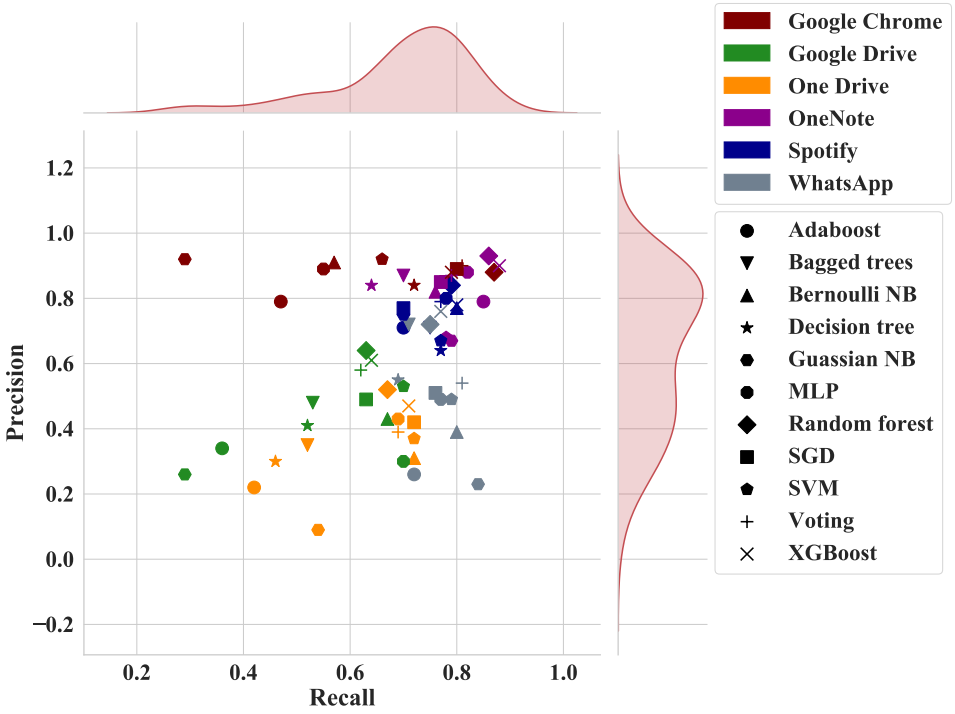

Figure 3.9 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of original data.

unknown dataset with padded packets. Bernoulli NB and XGBoost are more resilient to this masking method and their precision slightly decreases. On the other hand, the precision of the Voting classifier drops to about 0.49. The other ML algorithms experience a drop of up to 0.58 in their precision and recalls.

Figure 3.19 shows the distribution of precision and recalls of each application after packet length equalization on the unknown test dataset. The distribution of precision and recalls are right-skewed, showing that the performance of the ML algorithms decreases. Google Drive and WhatsApp are less detectable than the other applications with this masking method. However, Google Chrome and Spotify remain detectable.

Equalized packet count decreases precision and recalls more effectively than equalized packet length, as Figure 3.16 shows. Similar to the split dataset, Bernoulli NB and XGBoost are more resistant to this masking method than the other ML algorithms. However, Random Forest precision is 0.34 lower (from 0.91 to 0.58) in

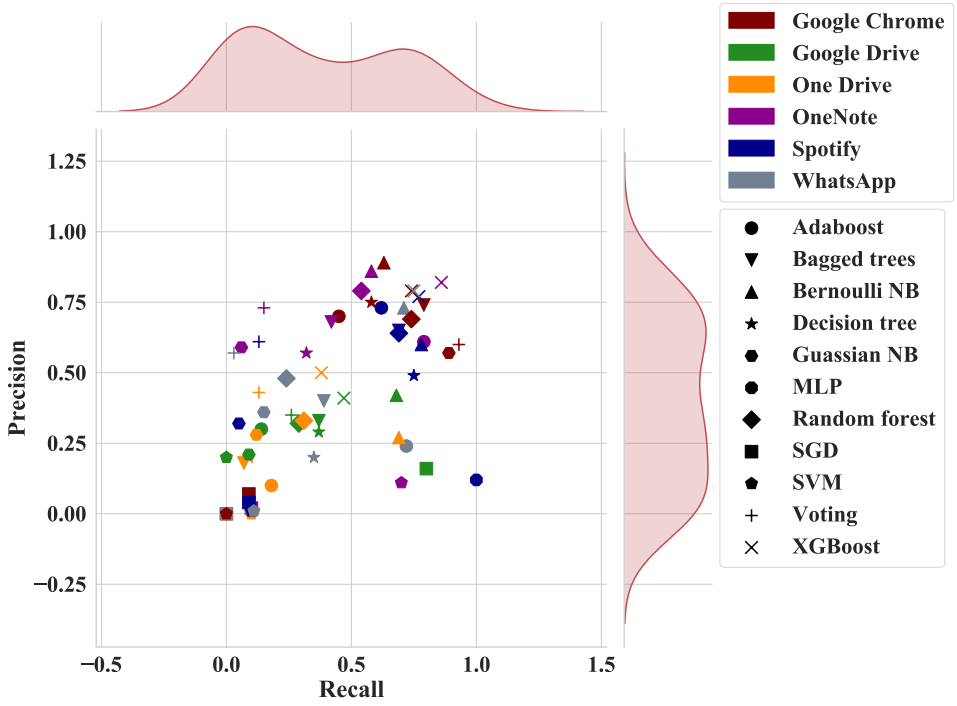

Figure 3.10 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of equalized packet length.

the original dataset. The most affected ML algorithms in this test are MLP and SVM, with a 0.56 decrease of precision.

Figure 3.20 shows the distribution of precision and recalls for all the applications, which are slightly right-skewed. Here, the precision and recalls of applications are spread in the graph, showing that this equalization method affects each application and that depends on the classifier used. For instance, Spotify has a precision close to zero when Adaboost is used. But Spotify has a precision close to 1.0, when XGBoost is used. Similar results are achieved for the other classifiers and applications.

Figure 3.17 shows the macro-averaged precision and recalls of the ML algorithms on the unknown dataset with equalized inter-arrival times. As this figure shows, this masking method is the least effective of the three, as in the split dataset test. Nevertheless, equalized inter-arrival times affects Bagged Trees more than the other classifiers, as it reduces its precision by about 0.22 (from 0.81 to 0.59).

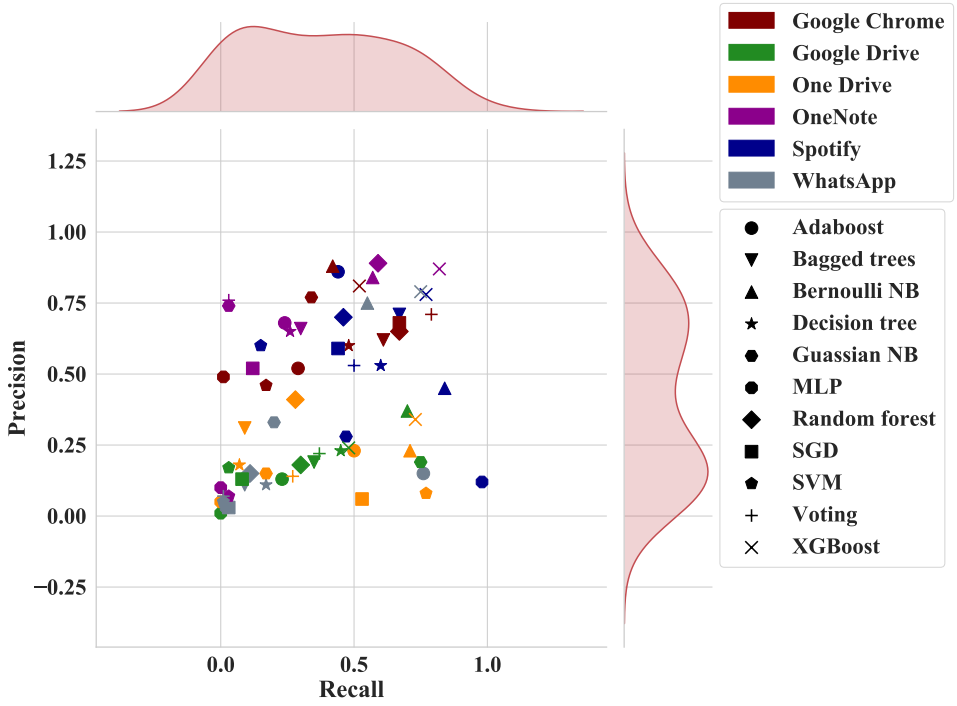

Figure 3.11 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of equalized packet count.

Figure 3.21 shows the distribution of precision and recalls for each application and ML algorithm when equalized inter-arrival times is applied. The results here show a left-skewed distribution for precision and recalls, meaning that this masking method is not as effective in decreasing the performance of the ML algorithms as the other methods. Precision of Spotify is close to zero when classified by SGD and Gaussian NB. Precision and recall of Google Drive are also close to zero when classified by Bagged Trees or Decision Tree. These results show that these ML algorithms use packet inter-arrival times as a discriminating feature. Therefore, this masking may be helpful in masking Spotify and Google Drive.

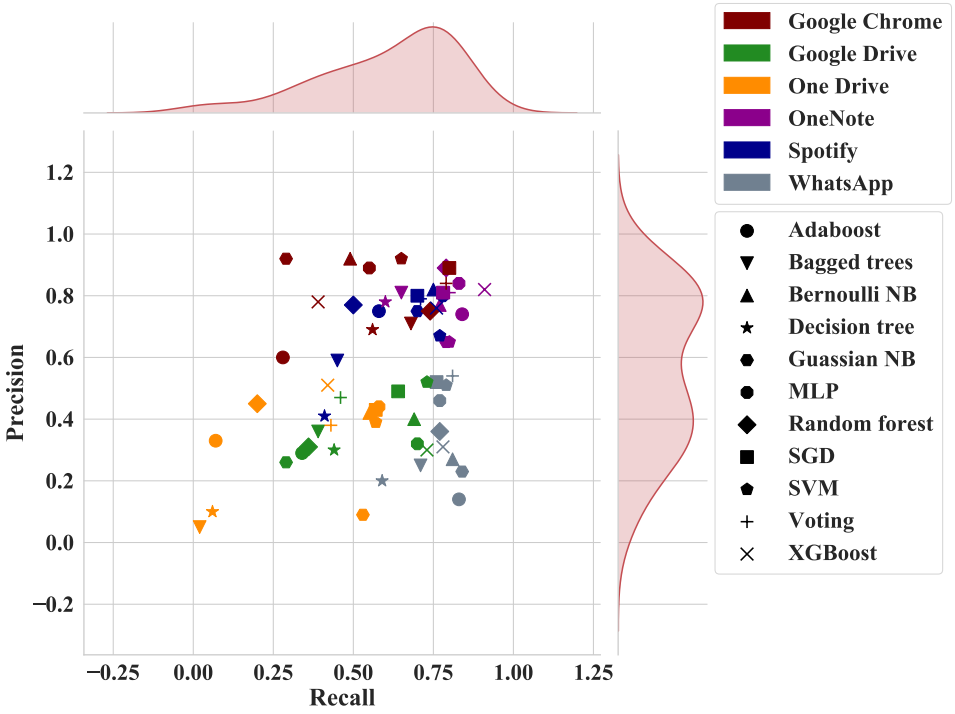

Figure 3.12 Precision vs. recall of ML algorithms on the split test dataset with kernel density estimations of equalized packet inter-arrival times.

Table 3.4 F1-score distributions stats on the unknown dataset

	Mean	Median	Mode	Skew
Original	0.711	0.78	0.78	-1.258
Equalized packet length	0.371	0.275	0	0.352
Equalized packet count	0.331	0.24	0	0.598
Equalized inter-arrival times	0.621	0.71	0.96	-0.799

Figure 3.22 compares the cumulative KDE of the masking methods based on the F1-score of the considered ML algorithms calculated for each application. Table 3.4 shows the statistics of the F1 score distributions. Here, we observe that the test results are similar to those obtained on the split dataset. Equalized packet count and equalized packet length significantly outperform equalized inter arrival time.

Figure 3.23 shows the confusion matrices of the studied ML algorithms for the masking methods. This figure shows that XGBoost, Voting, Random forest, and Bernoulli NB classify each application in the original dataset with few errors. With

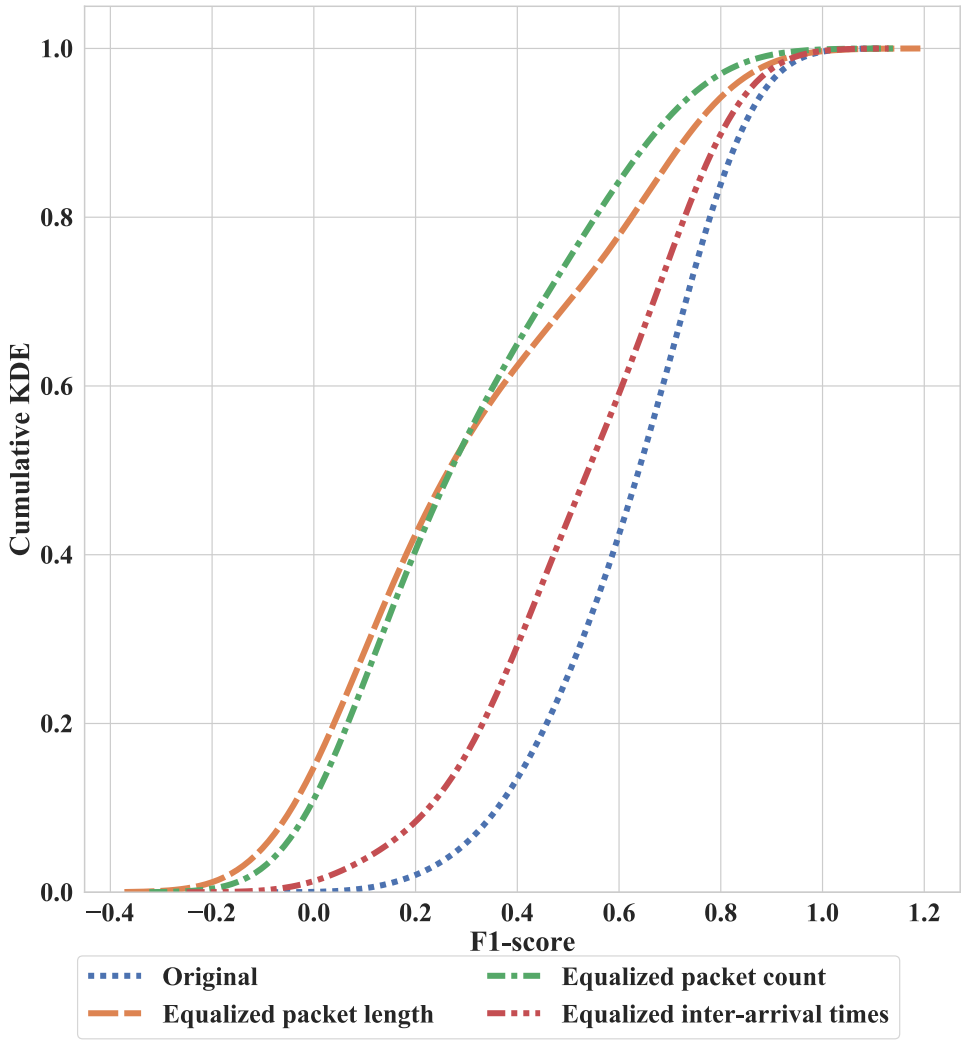

Figure 3.13 F1-score cumulative KDE of the masking methods on the split dataset.

equalized packet length masking, XGBoost, Adaboost, and Bernoulli NB are less affected than other ML algorithms. Bernoulli NB and XGBoost are more resilient against equalized packet count, but the other ML models make significant mistakes in their predictions. This figure also shows that the equalized inter-arrival times is the least effective masking method among the proposed methods.

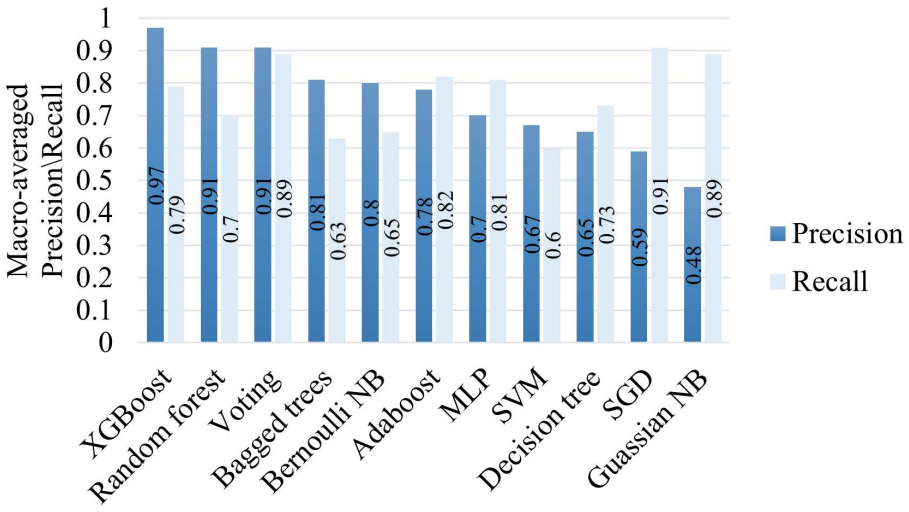

Figure 3.14 Macro-averaged precision of different ML algorithms on the unknown dataset of original data.

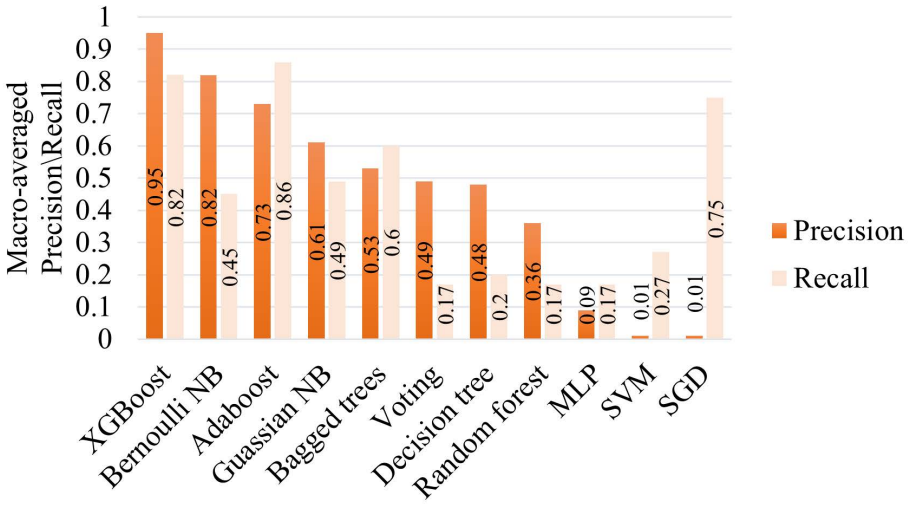

Figure 3.15 Macro-averaged precision of different ML algorithms on the unknown dataset of equalized packet length.

3.4 Discussion

3.4.1 Impact of Traffic Statistics Modifications

Modifying the considered traffic metrics may impact the transmission of data of a flow carried by the packets. Each metric equalization may affect the original traffic differently. The effect may be insignificant on some and may add some delays on others. Note that in an end-to-end flow, the sender performs the equalization

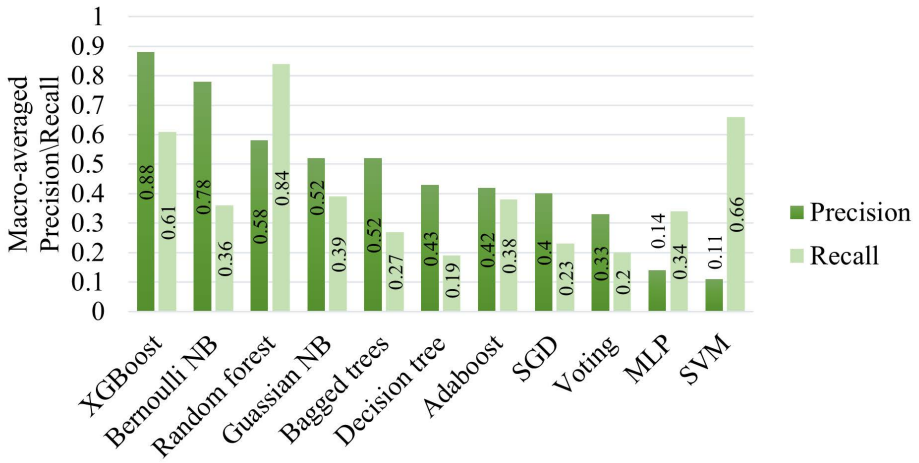

Figure 3.16 Macro-averaged precision of different ML algorithms on the unknown dataset of equalized packet count.

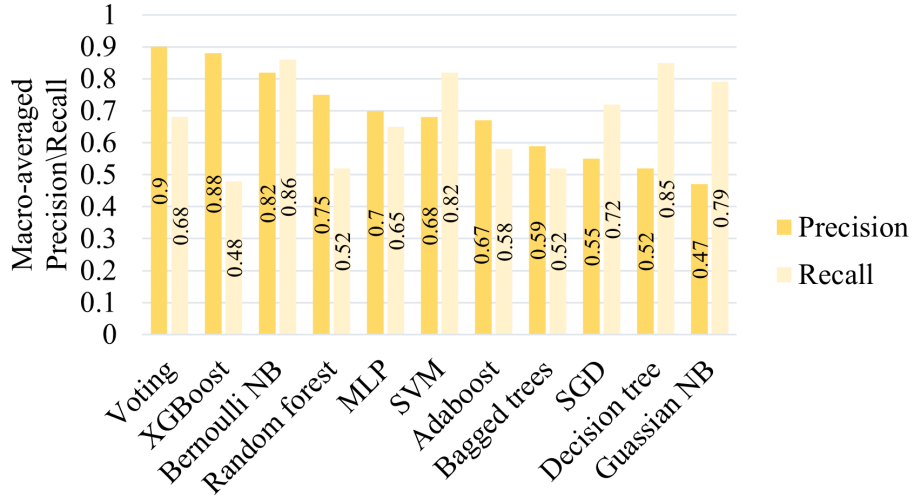

Figure 3.17 Macro-averaged precision of different ML algorithms on the unknown dataset of equalized inter-arrival times.

approach on the fly. This means that the sender may estimate the adjusted metric on a portion of the flow as it is being transmitted.

With equalizing packet length, a receiver may identify the data from the padding bytes by a pre-defined protocol between sender and receiver, for example, by using the random padding scheme in the Secure Shell (SSH) protocol [84]. In the worst case, this equalization method may add many bytes on a packet, and that may increase the transmission time of the modified packet and, sometimes, add some additional

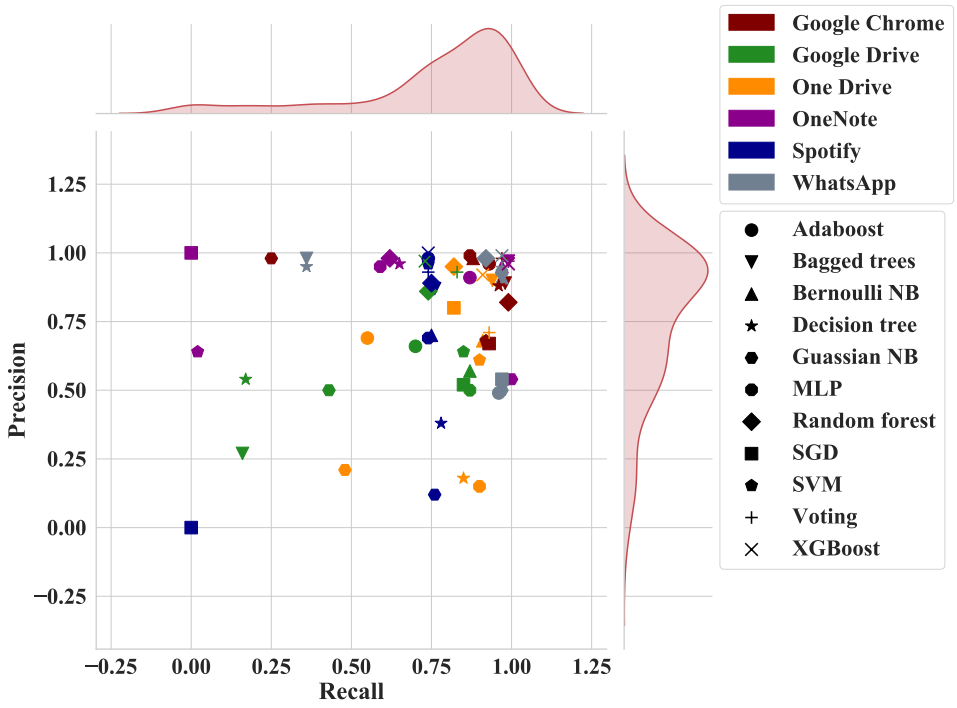

Figure 3.18 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on original data.

delay to the packet that follows. Whether such padding affects additional packets, it depends on the size of the inter-arrival times of the packets.

Equalization of packet counts requires the injection of dummy packets to flows on the fly and the removal of them from the actual data at their destination. The sender may consider packet rate to equalize the packet count on the fly. For that, the sender monitors the packet generation rate of a TCP flow and if the rate is not equal to the maximum, the sender injects new packets to the flow. To remove the dummy packets at the receiver, the encrypted payload may include a flag or use the IP header optional field to notify the receiver whether the packet is dummy or actual data. Adding dummy packets may cause a delay to one or more packets of the flow. The magnitude of that delay depends on when the injection of dummy packets takes place, the average packet length, and inter-arrival times. For example, consider a 1-second window of packet transmission with the original data rate of five packets per second. The sender may need to change the rate to 10 packets per second. If the

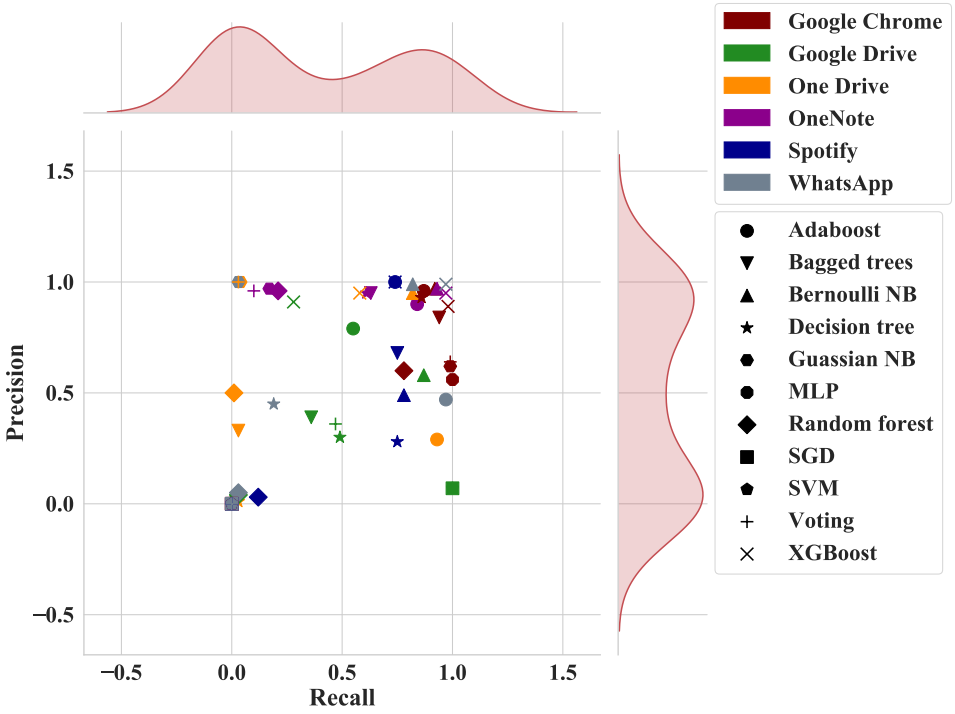

Figure 3.19 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on equalized packet length.

distribution of the generated packets in that second allows us to inject packets at the end of the 1-second window, our process may not delay any packet. However, if the sender injects dummy packets in between several data packets, then such injections may delay one or more of the original packets.

In equalized packet inter-arrival times, the sender holds the packets and transmits them in bursts with the minimum inter-arrival times. Holding the packets at the sender may cause delays on the transmission of packets if the holdout time is longer than the transmission time of the following packet, and so on. The advantage of this approach is that the receiver may not need to perform any additional operation on the incoming packets.

In summary, the use of these equalization techniques may need (a) to allocate resources to remove dummy bytes or packets at the receiver, and (b) sometimes, it may delay packets of a flow. Equalization of inter-arrival times is more prone to inject

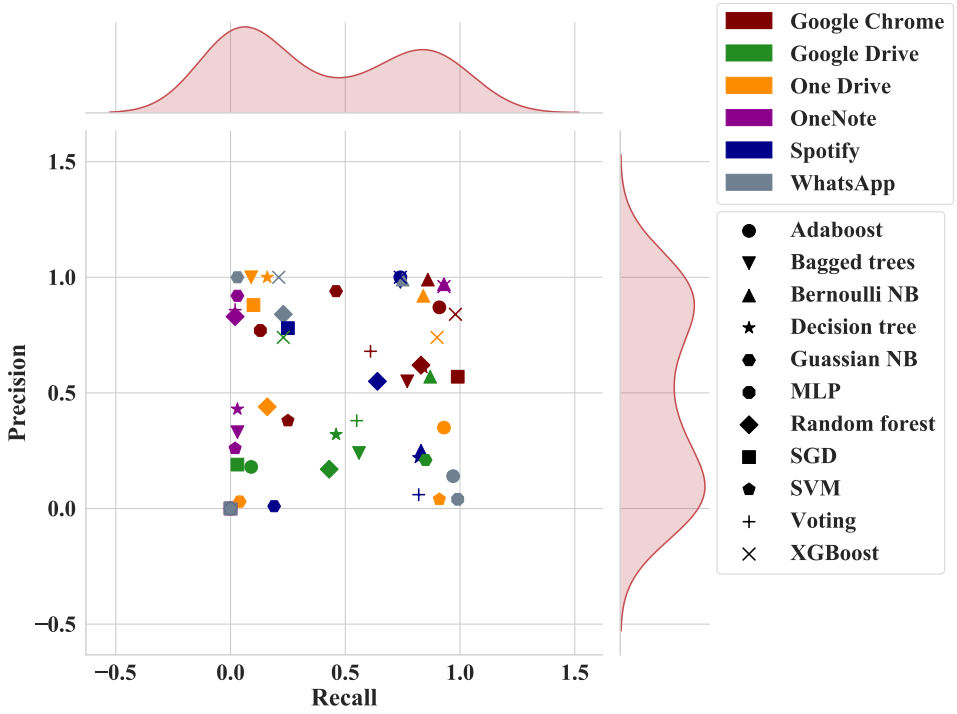

Figure 3.20 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on equalized packet count.

a transmission delay than the other two methods. However, this last method requires minimum or no support at the receiver side.

3.4.2 Comparison with Related Works

Table 3.5 compares our work with previous related works and their methods of countermeasure. Fu et al. [77] study the effectiveness of packet padding to counter statistical analysis on traffic such as sample mean, sample entropy, and sample variance. They show that packet padding can decrease detection rates to about 40%. Wright et al. [69] use a chi-squared test to specify the language of voice over IP (VoIP) as a binary classification, where they predict whether the language of VoIP is, for instance, English or not. They apply packet padding to VoIP packets to reduce the classification accuracy for about 27%. Chaddad et al. [78] propose using packet padding to decrease accuracy of traffic classifiers. The studied applications are Skype video call, Facebook browsing, 8 ball game, Whatsapp messaging, Viber VoIP, and

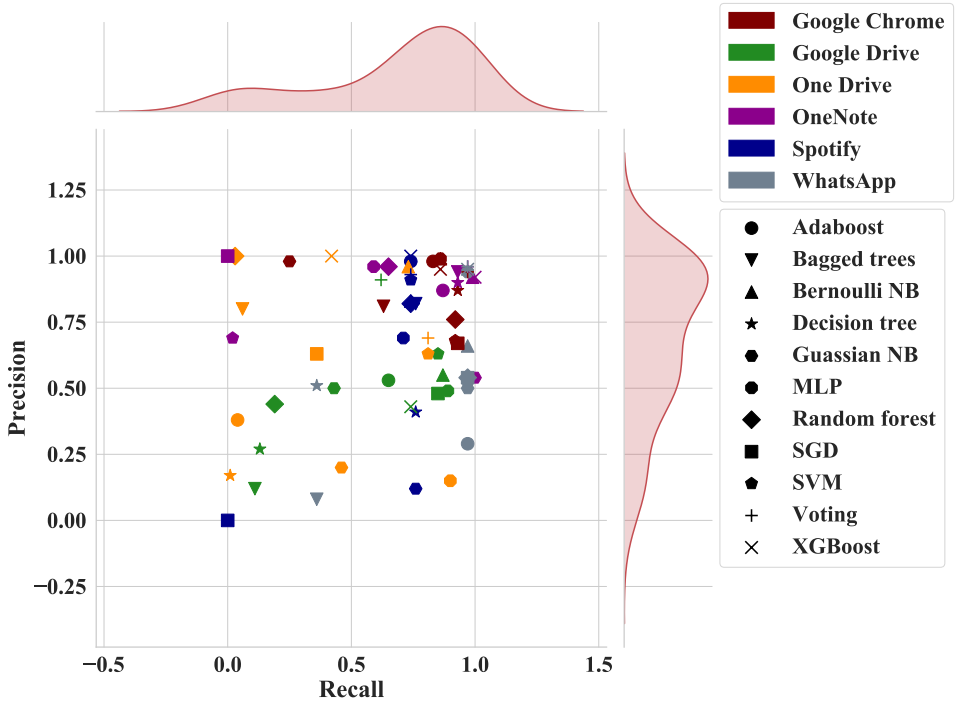

Figure 3.21 Precision vs. recall of ML algorithms on the unknown test dataset with kernel density estimations on equalized packet inter-arrival times.

Youtube streaming. Unlike our work, they study mobile application classification. And as only one application, Whatsapp, is shared between this study and ours, a direct comparison is not applicable. In our work, we achieve a success rate of 9 - 98.6% in reduction of ML algorithms precision with three different statistics equalization methods.

3.5 Conclusion

In this chapter, we showed how user applications are classified by adversarial ML algorithms through the analysis of the generated network traffic and showed how the traffic statistical features may be modified to counter the supervised ML classification. We showed that ML algorithms are capable of identifying user applications with precision up to 0.97.

We also used a new approach for labeling our datasets compared to previous works in the field. Instead of using port mapping, we used Microsoft event tracing to

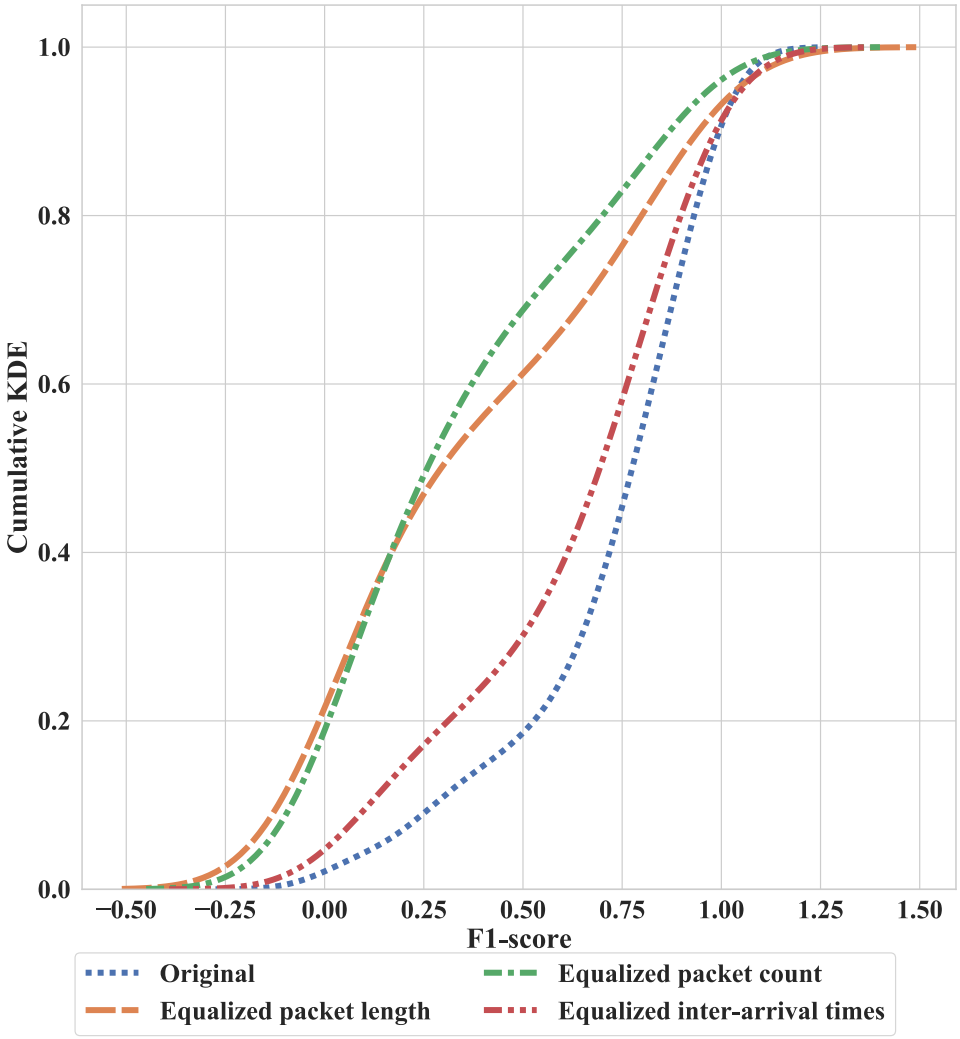

Figure 3.22 F1-score cumulative KDE of the masking methods on the unknown dataset for all three methods.

identify the applications generating the network traffic. In turn, it helped us identify the software used by online users instead of identifying packets as groups of services such as browsing and email traffic.

Among the studied classifiers, Voting and XGBoost perform better than the others when applied to split and unknown datasets. We also showed the performance of each ML algorithm in identifying specific applications. For instance, OneNote and Google Chrome achieve precision and recalls close to 1.0 when classified using Random Forest.

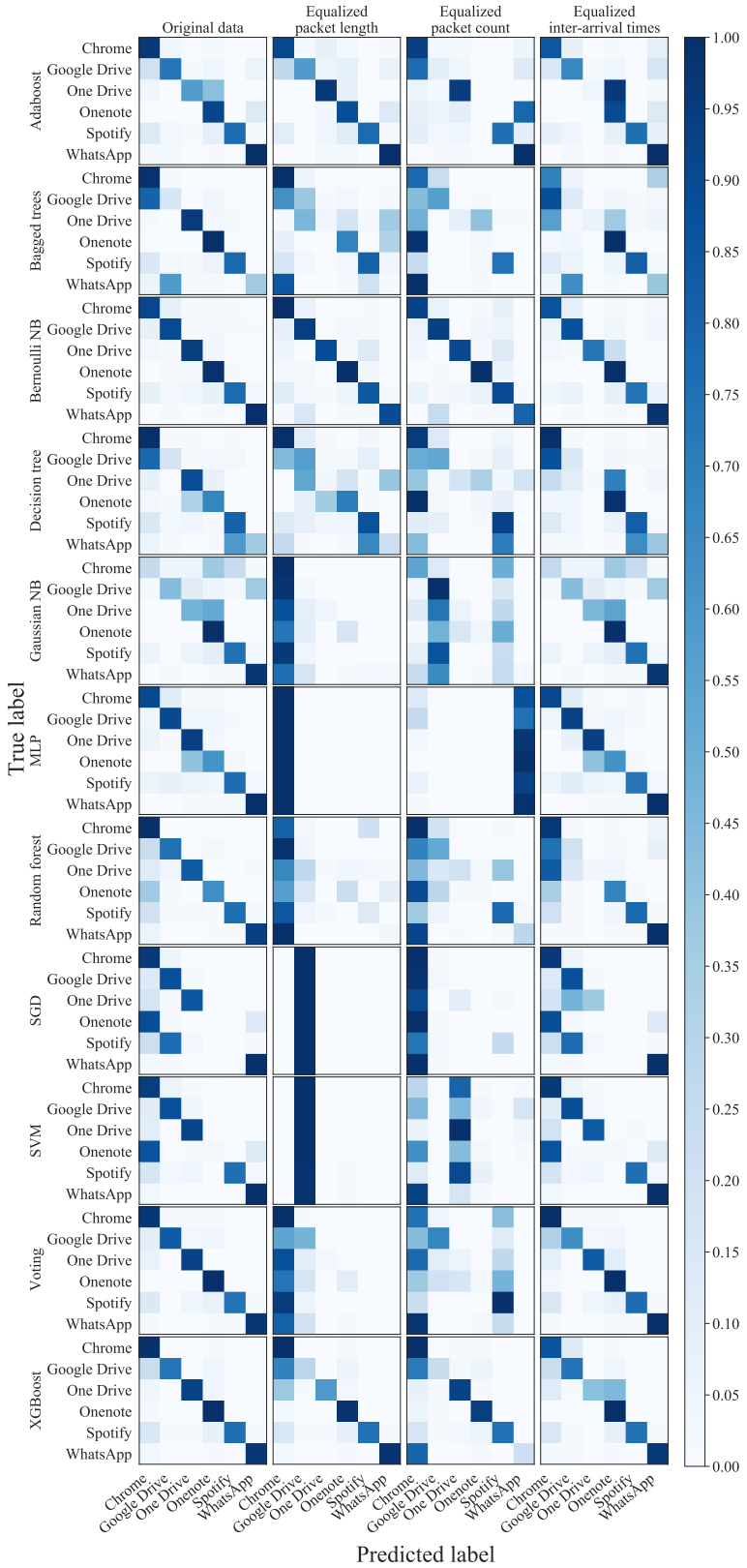

Figure 3.23 Confusion matrices.

To counter the effectiveness of ML algorithms in traffic classification, we have proposed using three masking methods: equalized packet length, packet count, and packet inter-arrival times. We chose these methods based on the distribution of traffic features for different applications. We showed that packet sizes, packet inter-arrival times and packet count are three discriminating features of network traffic. Therefore, we modified the statistics of these features to mask the traffic. We showed that equalized packet count is the most effective countermeasure of the three tested, as it decreases precision of ML algorithms from 0.09 to 0.56. The most single decrease in precision of one ML algorithm was 0.66 for SVM by using Equalized packet length. In our study, XGBoost and Bernoulli NB are the most resilient classifiers against the countermeasures, because their reliance on packet length, packet count, and inter-arrival times is less than the other tested ML algorithms. As a future work, we are planning to use deep learning models for these evaluations as well.

Table 3.5 Countermeasures against ITCs

References	**ML/Statistical Algorithms**	**Countermeasure**	**Measurement Variable**	**Success Rate (%)**
Wright et al. [69]	Chi-squared Test	Packet Padding	Accuracy	27
Dyer et al. [75]	Variable n-gram Naïve Bayes and SVM	Packet Padding	Accuracy	60
Fu et al. [76,77]	Bayes Classifier	Packet Padding	Precision	40
Chaddad et al. [78]*	SVM, Bagged-trees, KNN, and Random forest	Packet Padding	Accuracy	73 - 90
Our work	Adaboost, Bagged trees, Bernoulli NB, Decision tree, Guassian NB, MLP, Random forest, SGD, SVM, Voting, and XGBoost	Equalized packet length (packet padding), Equalized packet count, and Equalized inter-arrival times	Precision	9 - 98.6

*These works discuss mobile traffic classification.

CHAPTER 4

GAN TUNNEL: NETWORK TRAFFIC STEGANOGRAPHY BY USING GENERATIVE ADVERSARIAL NEURAL NETWORKS TO COUNTER INTERNET TRAFFIC CLASSIFIERS

In this chapter, we introduce a novel traffic masking method, called Generative Adversarial Network (GAN) tunnel, to protect the identity of applications that generate network traffic from classification by adversarial ITCs. Such ITCs have been used in the past for website fingerprinting and detection of network protocols. Their use is becoming more ubiquitous for inferring user information than before. ITCs based on machine learning can identify user applications by analyzing the statistical features of encrypted packets. Our proposed GAN tunnel generates traffic that mimics a decoy application and encapsulates actual user traffic in the GAN-generated traffic to prevent classification from adversarial ITCs. We show that the statistical distributions of the generated traffic features closely resemble those of the actual network traffic. Therefore, the actual user applications and any information associated with the user remain anonymous. We test the GAN tunnel traffic against high-performing ITCs, such as Random Forest and XGBoost, and we show that the GAN tunnel protects the identity of the source applications effectively.

A method used to undermine ITC classification is traffic obfuscation. This method bases its operation on performing packet padding and delaying transmission of packets to modify the traffic's profile [69, 75–77]. However, these methods are easy to circumvent because traffic modification focuses on a few and crude features. An adversary who uses a variety of ML algorithms can detect those changes as different ML algorithms are sensitive to different traffic features [2]. Therefore, there is a need for a method that can circumvent packet classification by obfuscating a holistic traffic profile.

To address this need, we propose using a generative adversarial network (GAN) model [53] to design a method that does not rely on modifying static traffic features to circumvent classification. Considering that GANs were originally used to generate fake images that are indistinguishable from real ones [53], we apply them but for traffic generation. In this chapter, we design a GAN that generates traffic flows that mimic the traffic generated by an actual application. This actual application, used as a decoy, can be a selectable one. We then use the generated flows to encapsulate and transmit the actual traffic with protected privacy against adversarial ITCs, as a tunnel. We call this approach GAN tunnel.

A GAN tunnel encapsulates actual network traffic and represents it as one generated by a different application, the decoy application. The traffic generated mimics the statistical characteristics of the decoy application. To test the efficiency of the proposed GAN tunnel, we use high performing ITCs, such as Random Forest (RF) [42] and XGBoost [65]. These ITCs aim to detect the source application of the transmitted traffic with high accuracy. We show that the ITCs classify the GAN tunnel traffic as belonging to the decoy applications.

4.1 Model Description

In our system, we create a tunnel between an online user and an application server, as Figure 4.1 shows. The GAN client and server embed actual network traffic into the generated traffic for the GAN tunnel. Here, the user sends its original traffic packets to a GAN client, which encapsulates them for transmission through an open network towards the destination. The GAN server extracts the packets from the GAN tunnel traffic and transmits them to the application server. The application server provides the service the users originally intended, such as transferring files from Google Drive. In the opposite direction and, similarly to the operation of the GAN client, packets originating from the application server are forwarded to the GAN server, and after encapsulation, the GAN tunnel server sends them to the GAN client. After that, the

GAN client extracts these packets and sends the decapsulated packets to the user's device.

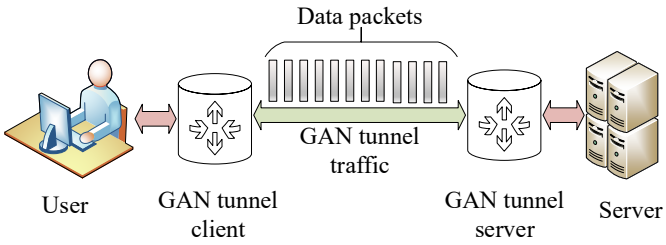

Figure 4.1 WGAN tunnel system overview.

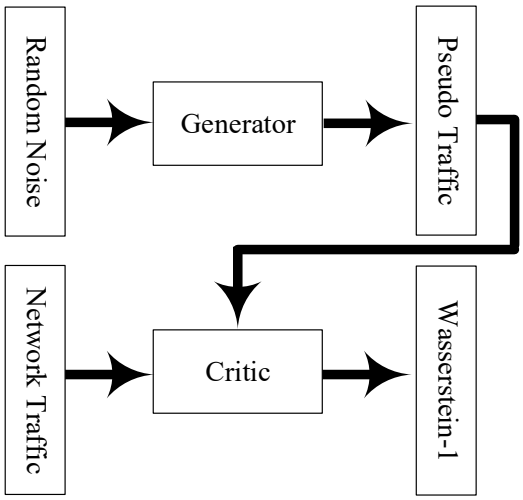

Figure 4.2 WGAN architecture adapted for network traffic generation.

4.1.1 Dataset Information and Preprocessing

In our analysis, we use the dataset from [2], which includes about 27,000 TCP flows. We define here a TCP flow as a stream of TCP packets exchanged between a client application and a server. These packets have the same source/destination addresses and port numbers in each direction of the flow [2]. For our WGAN model, we extract a feature set from the header fields of packets in TCP flows. Our feature set includes packet length, packet inter-arrival time, and TCP window size. Afterward, we construct new features that comprise total flow length, packet count,

and the minimum, maximum, median, mean, and variance of packet length, packet inter-arrival time, and TCP window size of each flow. We label the flows after the generating applications, namely Google Chrome, Google Drive, One Drive, OneNote, Spotify, and WhatsApp. Table 4.1 shows the distribution of flows per application. We scale the dataset features to the range (-1, 1) before training our WGAN by using the following normalization function [40]:

$$\frac{X - X_{\min}}{X_{\max} - X_{\min}}, \tag{4.1}$$

where X represents the values of one feature, and $X_{\min}$ and $X_{\max}$ are minimum and maximum of X. By fitting the features to be in the same range, the neural network produces results with smaller errors [85].

Table 4.1 Training Dataset Information before Balancing

	Number of records
Google Chrome	14,561
Google Drive	3,074
One Drive	756
OneNote	4,193
Spotify	3,179
WhatsApp	961

4.1.2 Flow Generation by Using WGAN

We build a generator network that outputs TCP flows with 17 constructed features. Table 4.2 shows the structure of the layers in our WGAN. The generator and critic models in our WGAN are multi-layer perceptron (MLP) neural networks with four and three hidden layers, respectively. We choose these numbers of hidden layers and their units by using the Random Search method [86]. In each hidden layer of

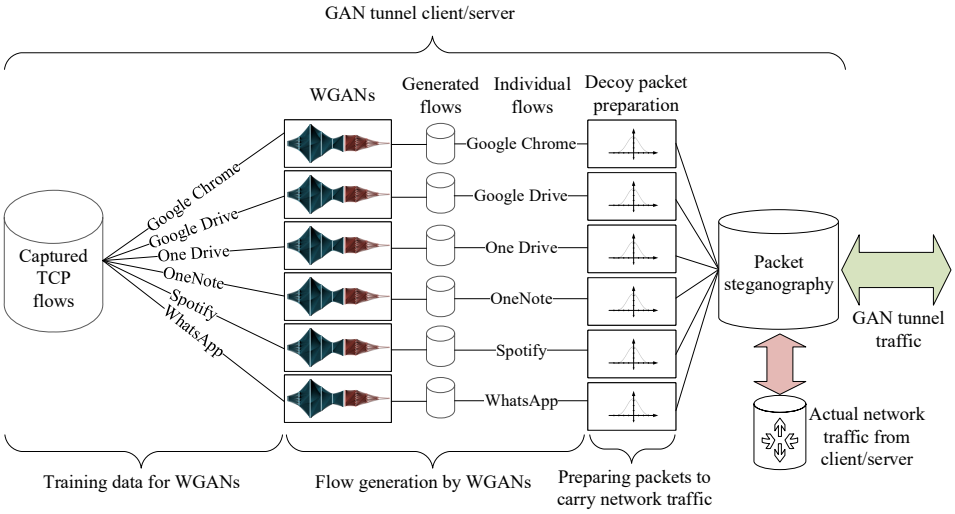

Figure 4.3 Sequence of Processes in the GAN Tunnel Client/Server.

the generator, we use the dropout regularization [87]. The dropout regularization prevents overfitting and improves the performance of neural networks [87]. In our WGAN, we use the dropout regularization to keep 80% of units active and to reduce the chance of co-adapting between units, and in turn, to create more robust features [87,88]. Dropping 20 and 50% of layer units is often found to be optimal for avoiding overfitting [87]. We also use the rectified linear unit (ReLU) and the Leaky ReLU as activation functions in the critic and the generator, respectively. The ReLU is the most commonly used activation function in neural networks [89, 90]. The ReLU assigns a zero gradient to neural network units with negative or zero inputs, and therefore, the units are deactivated with such inputs. Simply put, for an input value of x, the ReLU outputs $max(x, 0)$. We use Leaky ReLU functions in the generator layers. Unlike ReLU, the Leaky ReLU allows adding a small gradient to the units when they are inactive. That gradient improves the performance of the neural network by increasing sparsity and dispersion of hidden units activation [52]. The Leaky ReLU's output for an input value of x is αx for $x < 0$ and x otherwise, where α is a small slope coefficient for negative inputs. At the output of the generator, we use hyperbolic tangent (tanh) as the activation function to set outputs between (-1, 1),

Table 4.2 WGAN Design

		Number of Units	Regularization	Activation
Generator Layers	**Input**	100	-	-
	Layer 1	500	20% Dropout	Leaky ReLU
	Layer 2	2,000	20% Dropout	Leaky ReLU
	Layer 3	1,000	20% Dropout	Leaky ReLU
	Layer 4	500	20% Dropout	Leaky ReLU
	Output	17	-	tanh
Critic Layers	**Input**	17	-	-
	Layer 1	500	-	ReLU
	Layer 2	300	-	ReLU
	Layer 3	100	-	ReLU
	Output	1	-	Linear

similar to the scaled input, as discussed in Section 4.1.1. The output layer of the generator has 17 units, each representing a feature of the generated traffic.

In the GAN tunnel, we train a WGAN for each studied application, as Figure 4.3 shows. The captured TCP flows in our training dataset are divided by source applications to train each WGAN; one class per each WGAN. Each WGAN uses the design described in Table 4.2. Our system comprises six WGANs that are trained and operated in parallel. After training, the WGANs generate synthetic traffic flows, marked as flow generation stage by WGANs in this figure. The generated flows are then individually fed to a packet preparation module, which generates artificial packets based on the features of the flow. The packet preparation module calculates the packets' requirements, such as their average sizes and the time gap between them matching the flows' features. At the output of the packet preparation module, the information about the generated flows, such as the start and end of flows with TCP handshake packets, and information about data and acknowledgment packets are made available. After that, the detailed information about packets of the generated flow is fed to the packet steganography module, as Figure 4.3 shows. At this stage, a connection is created between the GAN tunnel client and server based on the input

information of this module. The payloads of the generated packets encapsulate the actual network traffic from client/server at this stage and before transmission. Here, we consider that packet payloads are encrypted. The encapsulated traffic is then transmitted over the tunnel whose flows resemble the decoy application's flows.

4.1.3 Packet Steganography and GAN Tunnel

This module generates the packets as described by the statistics that the WGAN outputs. Then, it encapsulates the packets of the actual traffic with the generated packets. The module generates the information and fields used by the tunnel protocol and the fitting of the packet lengths and inter-arrival gaps. It also provides reassembly information needed after performing segmentation, padding, or concatenation, and adds higher layer protocols needed for securing packet payloads.

As an example, the GAN tunnel may transmit traffic that resembles that originated by Google Chrome (i.e., the decoy application), which carries Spotify data (the actual traffic source). If this module segments the encapsulated traffic over several generated flows, it also creates a control connection between the GAN tunnel client and server by using one of those flows. The control connection carries information about the start and end packets of the segmented flows of actual traffic. The control connection uses transport layer security (TLS) to secure its traffic.

If the GAN tunnel uses TLS to transmit flows, the module performs the required encryption. Without TLS, the tunnel uses a control connection to exchange header encryption keys between the GAN tunnel client and server. We provide an example of packet encapsulation in Section 4.2.6.

4.2 Performance Evaluation

We run our WGAN on a PC with Microsoft Windows 10. This PC uses an Intel Core i7-8700K processor with 32 GB of memory and an NVIDIA Geforce GTX 1070 graphics card with 8 GB of dedicated memory. We use Keras 2.3 library [91] with

TensorFlow 2.0 backend [56] to test our WGAN designs, and Scikit-learn library [40] for ML classification.

4.2.1 Evaluation of Implemented WGANs

To investigate the performance of WGANs (one for each application), we check the values of Wasserstein-1 (1.13) between the actual and generated traffic at every training step. Figure 4.4 shows the calculated Wasserstein-1 values for each WGAN after applying a median filter similar to the one used in [54]. Here, the Wasserstein-1 values fluctuate between 0 and 0.03 in the first 2,000 training steps or epochs. An epoch is a cycle of training during which the WGANs go through all records of the training dataset. Afterward, the Wasserstein-1 values converge to almost zero (i.e., smaller than 0.002) for the six considered applications. We train the WGAN up to 10,000 steps because the Wasserstein-1 values are minimal and show enough convergence at this point. These results show that the distributions of the generated network traffic closely follow each of those of the actual traffic.

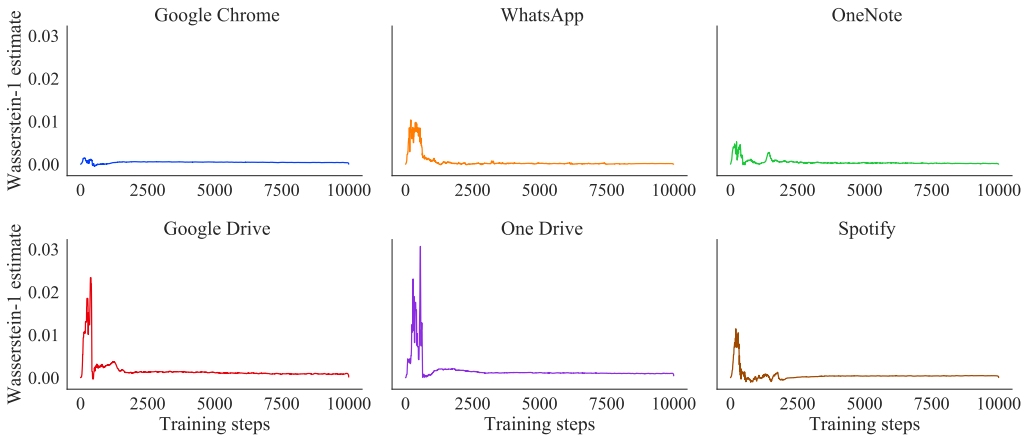

Figure 4.4 Wasserstein-1 values per application.

After calculating the Wasserstein-1 values as shown in Figure 4.4, we further evaluate the performance of our WGANs by comparing the distributions of generated features with those of actual network traffic. Figure 4.5 shows the kernel density

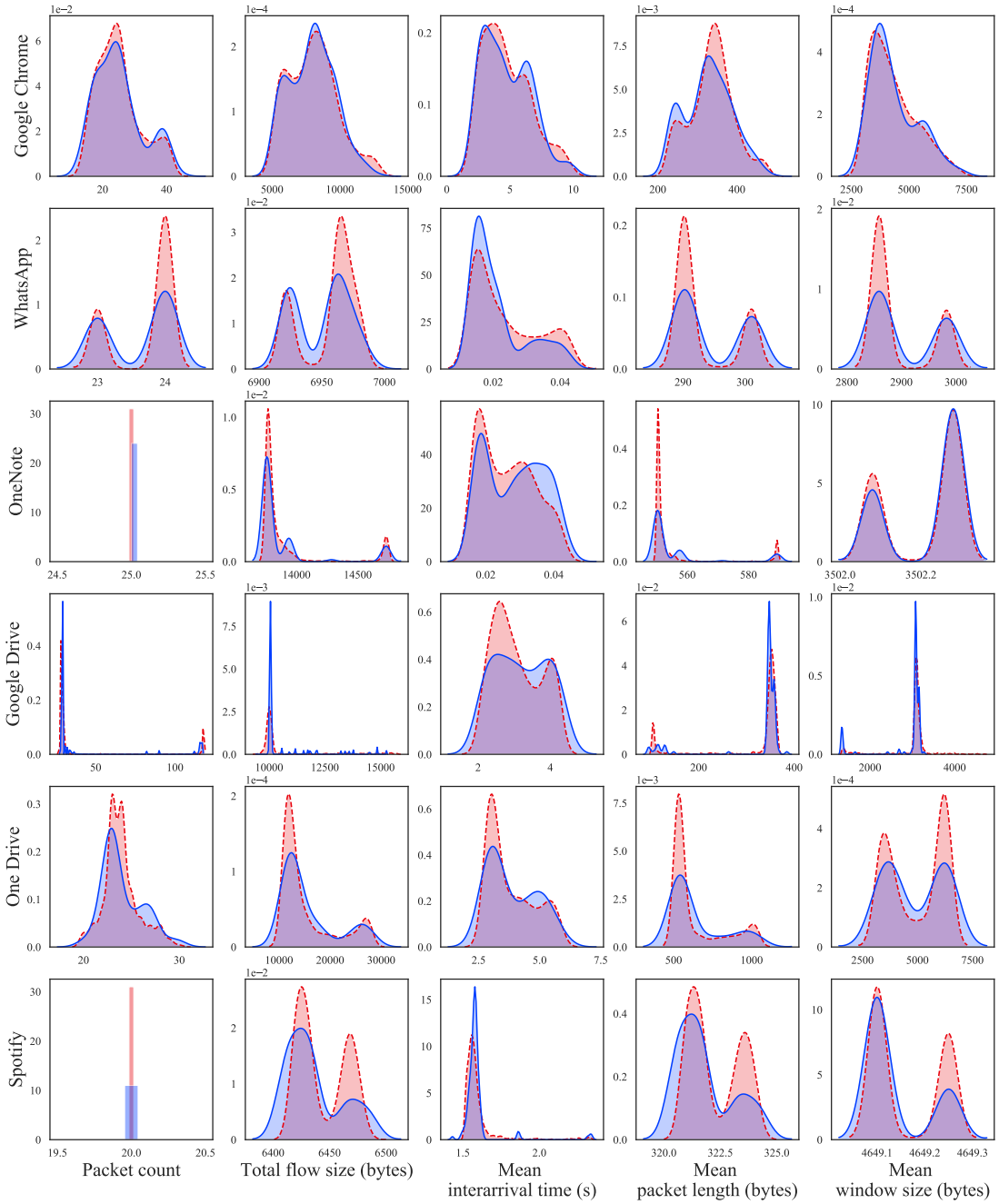

Figure 4.5 Kernel density estimations of generated and actual traffics' features for each application.

estimation (KDE) of the actual and generated network flows. In this figure, we include five features for comparison: packet count, total flow size, mean interarrival time, mean packet length on wire, and mean TCP window size. We show these features to compare the performance of the WGANs for the studied applications. This figure shows that the distribution on each of the traffic features of the WGAN generated traffic closely resembles that of the actual traffic.

As the graphs show, the generated and actual traffic have similar minimum and maximum values. The distributions of the generated packet counts for each of the considered applications are very similar to those of the actual network traffic. For instance, the most frequent number of packets per WhatsApp flow in the actual and generated traffic are 23 and 24, respectively. We also see such similarities in the packet count distributions of each of the considered applications. For instance, OneNote flows have a mean size of 25 packets per flow in both generated and actual traffic.

The number of packets per flow in each of the applications is also similar for the generated and actual traffic. For example, most of the actual Google Drive traffic flows have a total flow size of 10,000 bytes and have a global maximum in the distribution graph at this number. This figure also shows that the generated traffic distribution of Google Drive has local maxima at the same points as the actual traffic. This graph also shows that most of the flows in the actual and generated traffic have similar distributions of flow sizes.

Except for OneNote, other applications have similar minima and maxima in the actual and generated traffic with slightly different kurtosis in their mean interarrival times. Kurtosis is the fourth moment of a distribution that measures how much the data is spread in the tails of the data distribution [92]. We later show that the small discrepancy of OneNote interarrival times between the actual and generated traffic distributions does not affect the performance of the WGAN tunnel against adversarial ITCs.

As Figure 4.5 shows, the mean packet lengths of the generated and actual traffic for the considered applications have similar distributions. Here, for OneNote, we can see that the generated traffic is more widely spread around local maxima and have a smaller kurtosis than the actual traffic. Despite the differences, the generated and actual traffic have the same local maxima. The mean window sizes of the generated traffic for all applications correctly follow the distribution of those of the actual traffic. Here, the WhatsApp, One Drive, and Spotify WGANs generate more packets at local maxima. Therefore, they have a slightly higher peak value and a larger kurtosis than the actual traffic.

4.2.2 Time Complexity and Run Time

The time complexity of MLP networks with backpropagation is reported to be $O(PLn^2)$, where P is the dimension input data, L is the number of layers, and n is the number of units in each layer [93]. Here, $O(n^2)$ represents the matrix multiplication time complexity. With the parallelization of matrix multiplication, the time complexity for this function becomes $O(n)$. Subsequently, the time complexity of the MLP network becomes $O(PLn)$. However, with a large n, the processor may not have enough resources to perform the matrix calculations in $O(n)$ time. With parallelization on GPUs, the training time is significantly shorter than that for CPUs [94]. In our work, the time to generate flow features to avoid delays in the GAN tunnel operation is critical. After training our WGANs on the considered applications, we measured the time it takes to generate flow features. Table 4.3 shows the run time that a trained WGAN takes to generate 1,000 flows. Here, the average time to generate one flow is 7.03×10^{-5} s.

4.2.3 Evaluation of WGANs and ITCs Trained on the Same Dataset

To evaluate the performance of the GAN tunnel, we test whether the ITCs classify the generated flows as generated by the intended applications and not by the actual ones.

Table 4.3 Run Time to Generate 1,000 Flows

	Run time (s)
Google Chrome	0.071
Google Drive	0.075
One Drive	0.071
OneNote	0.069
Spotify	0.068
WhatsApp	0.068

In this test, we balance the actual data for each considered application by using a combination of *Synthetic Minority Over-sampling Technique* (SMOTE) [80] and Tomek-links under-sampling method [81] implemented by Imbalanced-learn library [82]. Balancing here refers to changing the number of records in each class (i.e., a category or label) so that they can have a similar number of records to other classes. In SMOTE, the minority class is over-sampled by taking each minority class sample and adding synthetic samples along the line segments joining any or all of the k minority class nearest neighbors [80, 95]. A minority class is a class with the smallest number of records. The combination of SMOTE and the under-sampling method yields more effective classification results [80]. In Tomek-links under-sampling, one of the two nearest samples is removed [81]. Moreover, to avoid overfitting the classifier on the training data, we use stratified 10-fold cross-validation for training the ML algorithms of ITCs. Data balancing is applied to the training portion of each cross-validation fold. After balancing, the number of training samples for each application is 12,000. The proportion of test samples in each fold follows the distribution of the original dataset (Table 4.1).

We train RF and XGBoost classifiers on this dataset and achieve 0.99 average precision and recall for every category of applications. These classifiers achieve high precision and recall in detecting network-flow patterns [2]. They achieve an average accuracy of 0.99 on training data folds. We then calculate the precision and recall

of the classifiers for the generated flows. To exhaustively test our generator, we generated a large number of samples for each application (about 1 million). With more generated samples, the probability of generating noisy samples is inevitable. In this way, we stress test our generator model to assess the quality of our results and ensure that the achieved precision and recall are justified. We found that regardless of the number of generated samples, our generated flows closely resemble actual network traffic.

Table 4.4 shows the classification results for the RF. Here, the table shows that generated traffic fully mimics the traffic of actual traffic. The generated flows for all applications are identified as authentic traffic of the decoy applications by a RF ITC in our tests. This result also shows that the actual application, whose packets are encapsulated by the GAN Tunnel, is not detectable by adversarial ITCs. Table 4.5 shows the classification results for the XGBoost classifier on the generated

Table 4.4 RF Results on Generated Data

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	1.0	1.0	1.0	998,900
Google Drive	1.0	1.0	1.0	907,200
One Drive	1.0	1.0	1.0	965,200
OneNote	1.0	1.0	1.0	805,100
Spotify	1.0	1.0	1.0	995,500
WhatsApp	1.0	1.0	1.0	970,700

flows. XGBoost is a boosting algorithm that calculates the final prediction based on error residuals of prior decision-trees classifiers [65]. XGBoost uses a gradient descent algorithm to minimize a loss function when adding a new decision trees (DT) classifier [65]. XGBoost is found to be a highly sensitive classifier [1]. Here, XGBoost achieves a near-perfect overall accuracy of 0.99. The precision of XGBoost falls to 0.98 for One drive, and its recall falls to 0.97 for Google Drive. These performance

losses are negligible, and the ML algorithm still detects the traffic as the decoy traffic with very high accuracy. Therefore, XGBoost is neutralized.

Table 4.5 XGBoost Results on Generated Data

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	1.0	1.0	1.0	998,900
Google Drive	1.0	0.97	0.99	907,200
One Drive	0.98	1.0	0.99	965,200
OneNote	1.0	1.0	1.0	805,100
Spotify	1.0	1.0	1.0	995,500
WhatsApp	1.0	1.0	1.0	970,700

4.2.4 Evaluation of WGANs and ITCs Trained on Different Datasets

Here, we demonstrate that the traffic that travels through the GAN tunnel remains anonymous, even when we train the WGANs on data that is not available for training the ITCs. For this, we split our original dataset into two parts, in a stratified manner. With the first part of the dataset, we train our WGANs, and with the second part, we train the ITCs. We balance the portion of the dataset to train ITCs using the combination of SMOTE and Tomek-links under-sampling. We also use a 10-fold stratified cross-validation. The accuracies of both RF and XGBoost on training dataset folds are, on average, 0.99; similar to the previous test.

Table 4.6 shows the results of a RF ITC on the generated flows based on the split dataset. Here, the precision on Google Chrome is slightly lower than that in the previous test. The achieved precision means that this ITC detects a few false positives in its classification. The recall of Google Drive becomes 0.84, which is lower than that of the previous test. This performance decrease shows that the RF ITC detects false negatives in the test results. Overall, with an average accuracy, precision, and recall of 0.97, 0.98, and 0.98, respectively, adversaries who use the traffic passing through

the GAN tunnel cannot identify the actual application that generated the traffic, and the private information it conveys.

Table 4.6 RF Results on Generated Data by using Split Dataset

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	0.87	1.0	0.93	999,300
Google Drive	1.0	0.84	0.91	970,700
One Drive	1.0	1.0	1.0	960,900
OneNote	1.0	1.0	1.0	994,800
Spotify	1.0	1.0	1.0	986,400
WhatsApp	1.0	1.0	1.0	974,700

Table 4.7 shows the precision, recall, and F-1 score of XGBoost on the generated flows based on the split dataset. Similar to the results of RF, the performance of XGBoost decreases in this test as compared to those where WGANs and ITCs are trained on the same dataset. The average accuracy, precision, and recall of XGBoost in this test are 0.97, 0.98, and 0.98, respectively, and the actual network traffic the GAN tunnel transports remains anonymous.

Table 4.7 XGBoost Results on Generated Data by using Split Dataset

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	0.85	1.0	0.92	999,300
Google Drive	1.0	0.82	0.90	970,700
One Drive	1.0	1.0	1.0	960,900
OneNote	1.0	1.0	1.0	994,800
Spotify	1.0	1.0	1.0	986,400
WhatsApp	1.0	1.0	1.0	974,700

4.2.5 Parameter Tuning of WGAN

As mentioned in Section 4.1.2, we use Random Search to choose the configuration of the WGANs. Table 4.8 lists the range of parameters for our search. We test the generated samples from the WGAN with selected parameters by RF using the split dataset. Table 4.9 lists some of the different selected parameters for WGANs. Figure 4.6 shows the F-1 scores of the RF classifier on generated samples from these WGANs. As the figure shows, WGAN 6 achieves the highest overall F-1 score among the tested designs. Therefore, WGAN 6 is our design choice.

Table 4.8 Range of Parameters of WGANs Designs

Generator hidden layers range	2 - 4
Number of generator units	10, 20, 50, 100, 200, 300, 1,000, 2,000
Generator hidden layers activations	Linear, ReLU, Leaky ReLU
Generator hidden layers regularizations	None, Dropout
Critic hidden layers range	2 - 3
Number of critic units	10, 20, 30, 50, 100, 300, 400, 500

4.2.6 Packet Generation from WGAN Flows

To create the encapsulating packets, we use the flow information generated by the WGAN: the number of packets, flow length, and descriptive statistics of packet lengths, packet interarrival times, and window sizes. For instance, we consider a generated Google Drive flow with the features listed in Table 4.10. Here, we can encrypt the header information of another application, such as Spotify, and encapsulate the entire packet (header and payload) as payloads of the packets of generated packets. In this case, the packet preparation module (Figure 4.3) reserves six packets for TCP handshakes of the generated flows for data transmission between

Table 4.9 WGANs With Different Parameters

	Generator hidden layers units	Generator activations/ regularizations
WGAN 1	20, 10, 5	Linear/None
WGAN 2	20, 50, 100	Linear/None
WGAN 3	100, 200, 300, 1000	Linear/None
WGAN 4	10, 100, 200, 500	Linear/None
WGAN 5	10, 1,000, 200, 500	Leaky ReLU/ None
WGAN 6	500, 2,000, 1,000, 500	Leaky ReLU/ Dropout
WGAN 7	500, 3,000 , 2,000, 1,000	Leaky ReLU/ Dropout
	Critic hidden layers units	Critic activations
WGAN 1	10, 20	ReLU
WGAN 2	10, 30	
WGAN 3	30, 50, 100	
WGAN 4	10, 30, 50	
WGAN 5	10, 300, 500	
WGAN 6	500, 300, 100	
WGAN 7	500, 400, 300	

the GAN tunnel server and client. Let us assume that the first two handshake packets are 66 bytes on wire, and the third one is 54 bytes. Therefore, a total of 372 bytes are subtracted from the total flow length (10,179 bytes), and the remaining 9,807 bytes are used for data transmission. With 23 remaining packets and an average packet length of 352 bytes, the GAN client/server sends packets that have lengths close to the average packet length, except for at least one packet with a size of 1,440 bytes to meet the maximum packet size of the generated flow as Table 4.10 shows. The interarrival times are, on average, close to 2.5 s (Table 4.10); therefore, the GAN client/server sends the packets at a fast rate to approach the mean. For the advertised window sizes, we use descriptive statistics of the generated flow to include

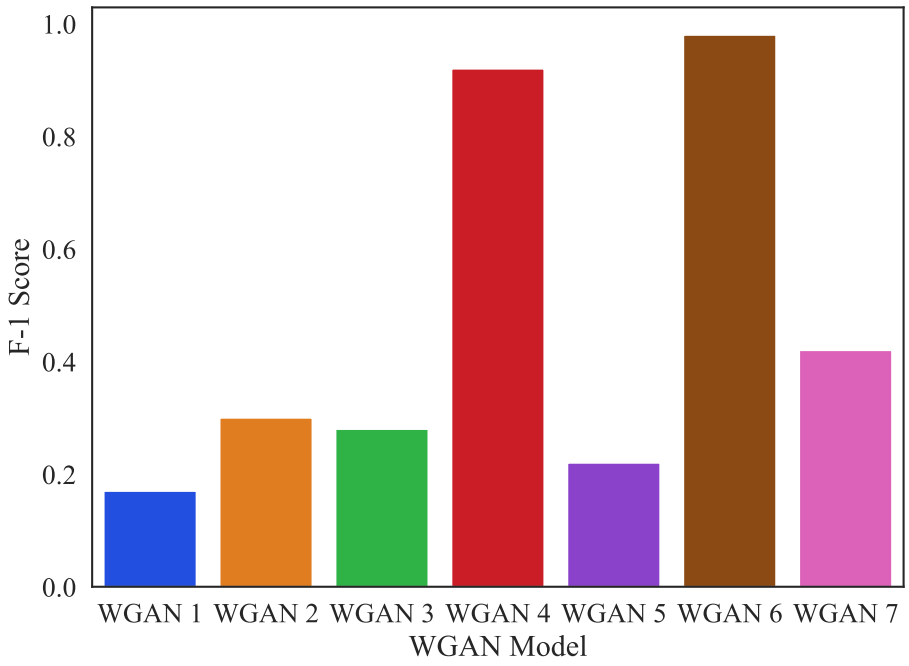

Figure 4.6 F-1 scores of RF on different WGANs. WGAN 6 achieves the highest F-1 score

minimum, maximum, and mean values in the exchanged packets. The median value of window sizes determines the number of packets used for the smallest and largest window sizes of the WGAN generated traffic. For instance, if the average window size is larger than its median (as Table 4.10 shows), the distribution of the window sizes are skewed to the right, and therefore, we have more packets with window sizes close to the smallest size. Table 4.11 shows a break-down example of a generated flow of Google Drive (included in Table 4.10) without using TLS.

With TLS between the GAN tunnel client and server, additional packets are required for TLS establishment. Therefore, we have fewer packets and a smaller number of payload bytes for encapsulating actual traffic. Following the same example of a generated Google Drive flow, as shown in Table 4.10, from 29 packets, the packet preparation module excludes six packets for connection establishment of TCP and TLS, and three packets for connection termination [96]. The key exchange between a TLS server and client also uses about ten packets based on observations of our

Table 4.10 A sample of a synthetic flow of Google Drive

Feature	Value	Feature	Value
Packet count	29	Average packet length	352.2117 bytes
Total flow length	10,179 bytes	Median packet length	91.9119 bytes
Min interarrival time	$9.999\times10^{-}4$ s	Variance packet length	234,802.5781 $bytes^2$
Max interarrival time	71.2822 s	Min window size	256 bytes
Average interarrival time	2.5568 s	Max window size	64,241 bytes
Median interarrival time	0.02227 s	Average window size	3,081.2285 bytes
Variance interarrival time	165.7221 s^2	Median window size	1,007.7315 bytes
Min packet length	54 bytes	Variance window size	136,675,952 $bytes^2$
Max packet length	1,439 bytes		

experimental traces. Therefore, the system has ten packets for data transfer between the GAN tunnel client and server. The payload of these data packets carries actual packets from other applications, such as packets from WhatsApp. The first two TCP handshake packets are 66-byte long, and the last one is 54 bytes, which satisfies the minimum packet length of the flow. A client TLS hello packet may have different sizes. In this example, we consider a 200-byte TLS hello packet. The size of the client hello packet depends on the used cipher suite. An acknowledgment packet from the server follows the hello packet, with a size of 60 bytes. The certificates exchange between server and client may differ slightly in size. A self-signed certificate is about 800 bytes. Here, we assume the use of 1,500 bytes per certificate. Overall, the TLS establishment and termination may require about 6,000 bytes, which are about half of the total length of the generated flow. Therefore, the GAN client/server can only

Table 4.11 Overview of Packets Needed for Handshakes and Data in a Generated Flow by our WGAN without using TLS

Packet Type	Count	Required Length (bytes)
TCP handshake	6	372
Data (max size)	1	1,440
Data	22	8,367

deliver about 4,000 bytes (Table 4.10) encapsulated data in this flow. From the remaining ten packets, at least one has the maximum packet length unless the TLS certificate already has satisfied this length. Moreover, those remaining packets use the average packet length. The interarrival times and advertised window sizes follow the same principles as described for TCP flows without TLS. Table 4.12 shows a sample of flow break down in the number of packets for a WGAN-generated flow when we apply TLS to GAN tunnel traffic.

Table 4.12 Overview of Packets Needed for Handshakes and Data in a Generated Flow by our WGAN with TLS

Packet Type	Count	Required Length (bytes)
TCP handshake	19	6,000
Data (max size)	1	1,440
Data	9	2,560

If the size of the data that a generated flow encapsulates exceeds the byte limit of that flow, the GAN client/server may either pick another and more suitable generated flow, or segments the data and transmit the segments by using more than one flow.

4.3 Background and Related works

Network traffic generation using GANs has been considered for detecting intrusion, dataset augmentation, or illegality detection [11, 97, 98]. Ring et al. [99] introduced flow-based traffic generation using GAN. This work aims to generate realistic network traffic to aid intrusion detection systems (IDS). The challenge that current intrusion detection systems face is the availability of labeled data and the high cost of false

positives [99]. Charlier et al. [97] used GAN to generate DDoS traffic to improve the accuracy of ITCs in detecting such an attack.

Network traffic data is mostly unbalanced among different categories. For instance, some applications may generate more packets than others. This feature of traffic affects the efficiency of ML-based ITCs as they may tend to favor a category of traffic when performing classification, and that leads the ITC to having biased decisions. To avoid such a problem, balancing methods, such as oversampling the minority categories (i.e., categories with the smallest number of records) or undersampling the majority categories, are employed for classification [80]. There are also advanced techniques that combine oversampling and undersampling methods [79] for data balancing and to achieve higher classification accuracy. An example of this is the combination of SMOTE and Tomek-links under-sampling method, as adopted in this chapter. Vu et al. [98] used a GAN to generate data for minority categories and augmenting the dataset to create a balanced dataset for classification by ML algorithms. This approach showed that ML classifiers can achieve more effective results in situations when sampling methods underperform.

ITCs may be used for website fingerprinting, which is the process of identifying websites visited by online users. Internet censorship may use such ITCs to identify and censor specific types of traffic. Li et al. [11] use GAN to generate features of uncensored traffic and morph the features of censored traffic, to avoid censorship. In this way, censored traffic looks as uncensored traffic. The authors change the features of traffic that goes to *sp0.baidu.com* to those of the traffic that goes to *www.baidu.com*. However, details on the implementation of the system are missing. In our work, rather than just focusing on packet count, which would be easily detectable by a ML classifier, we argue that many traffic features must be considered instead for an effective classification countermeasure. In fact, we use 17 constructed features to generate highly realistic flows, rather than features that do not represent the flows

Table 4.13 Comparison of objectives of security systems employing GAN

References	Objective	Defensive / Offensive
Ring et al. [99]	Dataset augmentation to aid IDS	Defensive
Hu et al. [100]	Mask API calls of malware to evade antivirus software	Offensive
Lin et al. [101]	Generating malicious traffic that evades IDS	Offensive
Rigaki et al. [102]	Modify malware network traffic to resemble Facebook chat traffic and evade IDS	Offensive
Cheng et al. [103]	Generation of realistic ICMP pings, DNS queries, and HTTP requests	N/A
Taheri et al. [104]	Dataset augmentation to aid anti-label flipping systems	Defensive
Li et al. [11]	Avoid censorship	Defensive
Our work	Privacy protection against adversarial ITCs	Defensive

in their entirety. Moreover, we propose that the generated flows carry the actual network traffic as payloads (a tunnel).

Table 4.13 compares previous works concerning computer and network security systems that use GAN. Here, we describe them with more detail and indicate the difference with the approach proposed in this chapter. Cheng et al. [103] use a GAN model to generate network traffic at the network layer. The generated traffic consists of Internet control message protocol (ICMP) packets, domain name system (DNS) queries, and HTTP web requests. Unlike this work, our GAN is trained to generate TCP flow attributes to create a GAN tunnel for data transmission. Our main goal is to counter adversarial ITC. Rigaki et al. [102] and Lin et al. [101] proposed a GAN model that intrudes a network and evades network intrusion detection systems. Unlike these works, we use a GAN model to create a secure communication tunnel that online users can employ to protect the identity of the application they use. Hu et al. [100] propose a GAN model to change the API calls of malware applications

Table 4.14 Comparison between architecture of security systems employing GAN

References	GAN Architecture	Test Mechanism	Main Measurement Variable	Success Rate (%)
[99]	MLP	Domain knowledge test	Ratio of correctly generated flows/ all generated flows	96-99% *
[100]	MLP	RF, SVM, DT, MLP, LR, and Voting	True positive rate	93-95%
[101]	MLP	RF, SVM, DT, MLP, NB, LR, and KNN	True positive rate	29-100%
[102]	LSTM	Stratosphere Linux IPS [105]	Ratio of successful/generated flows	55-90% *
[103]	CNN	Experimental TCP/UDP test	Ratio of successful/generated flows	66-99% *
[104]	CNN	RF, SVM, DT, MLP, and CNN	Average F-1 score	5-85%
[11]	MLP	SVM, NB, and Jaccard's coefficient	Area under receiver operating characteristic	2-88%
Our work	MLP	RF and XGBoost	Average F-1 score	97-98%

*Success rate is calculated based on number of flows.

that look like normal API calls to evade antivirus software. However, this model is only related to computer system security.

Table 4.14 compares the GAN architecture and test mechanisms of the works presented in Table 4.13. Depending on the type of features that we seek to generate, different types of GAN may enhance the generation of new traffic. For instance, Cheng et al. [103] generate network traffic at the byte-level while others generate traffic at the packet and flow levels.

As shown in this table, the test mechanisms considered in these works are domain knowledge test, ML algorithms, and experimental tests. Ring et al. [99] test generated flows based on the domain knowledge, such as checking that UDP packets do not include TCP flags. They measure their success rate by calculating the ratio of correctly generated flows over all generated flows and they achieve 96 to 99% success. Hu et al. [100] test the effectiveness of their model by using ML algorithms to classify their generated application programming interface (API) calls as non-malicious. The ML algorithms used are: RF, SVM, DT, MLP, logistic regression (LR) [106], and a voting classifier [107]. In this work, 93 to 95% of the generated API calls is classified as non-malicious. Lin et al. [101] also use ML algorithms as their test mechanism to check how much of the generated traffic is classified as normal. They use RF, SVM, DT, MLP, LR, Naive Bayes (NB) [46], and K-nearest neighbors (KNN) [95]. They achieve a true positive rate that ranges from 29 to 100%. Rigaki et al. [102] use a third-party software named Stratosphere Linux IPS [105] to find out the proportion of the generated flows that passes as normal traffic through this intrusion prevention software. Here, the success rate is measured by the ratio of successfully generated flows over the total generated ones resulting in 55 to 90% success rate. Cheng et al. [103] test the generated packets by sending them out as actual network packets and test them by observing if they receive a successful response. In this test, the packet generation achieves 66 to 99% success. Li et al. [11] measure the success rate of their model by using SVM, NB, and Jaccard's coefficient [108] to classify the

generated traffic. The measurement variable they have used is the area under operator characteristic (true positive rate vs. false positive rate) curve. With results closer to 0.5, the classifier is unable to identify the generated traffic from uncensored traffic and they achieve results in the range of 0.56 to 0.99 (2-88% success). Taheri et al. measure the effectivity of their anti-label flipping mechanism by using ML algorithms. They achieved average F-1 scores that range from 5 to 85% in defeating data flipping systems. In our work, we use XGBoost and RF to test the detectability of the generated flows. XGBoost is a robust tool to classify network traffic [2]. However, other works in Table 4.14 have not considered XGBoost for testing.

In our tests, we showed that the generated flows are identified as decoy application traffic with average F-1 scores of 97 and 98%. Compared to previous works that focus on network traffic generation [99, 101–103], the range of our success rate is more effective because it has smaller variability.

Convolutional neural networks (CNN) are a good choice for byte-level traffic generation because they are mainly used in image classification, where the input data are formed from vectors of RGB values for each image pixel. Another choice for traffic generation is long short-term memory (LSTM) [61, 102]. LSTMs are mainly used to model time-series data, where the model benefits from both historical and recent data. In other words, LSTM aims to learn the time evolution of sequences of data [109]. Such a model may be fit to generate a user datagram protocol (UDP) stream of data as UDP streams are not divided into flows of traffic. In our work, we use a fully connected network or MLP. In such a network, each unit of each layer is connected to all the units in the next layer. Each layer of a fully connected network represents a different transformation of data in feature space. Functions, such as Fourier and Laplace transforms have been used to simplify complicated equations to a suitable format for analysis. Similarly, a fully connected network is helpful in finding a suitable transformed data representation to simplify tasks of classification

or data generation [109]. Therefore, to represent a vector of input noise as flows in our dataset, we use MLP.

Other previous works resort to non-dynamic methods of countering ITCs. Fu et al. [76] analyze packet padding and delay to counter ping-probing attacks, which are used to identify user traffic rate based on ping round-trip time. The authors concluded that randomly delaying the packets is effective in countering ping probing attacks. This work, however, does not consider ML-based ITCs. Fu et al. [77] studied the effectiveness of packet padding to counter statistical analysis on traffic such as sample mean, sample entropy, and sample variance. They show that packet padding can decrease detection rates to about 40%. Wright et al. [69] used the chi-squared test to specify the language spoken during a voice over IP (VoIP) call as a binary classification. For instance, they predict whether the language used in a VoIP call is English or not. They applied packet padding to VoIP packets to reduce the classification accuracy to about 27%.

4.4 Conclusion

In this chapter, we introduced a GAN tunnel to counter adversarial ITCs. Our proposed GAN tunnel uses generated network traffic that mimics actual network traffic to avoid identification/classification of the originating application by adversarial ITCs. We designed a Wasserstein GAN (WGAN) with a generator comprising four hidden layers and a critic with three hidden layers. The WGAN uses traffic from original user applications for modeling the traffic of decoy applications. In this way, a user may configure the WGAN to generate traffic of a selectable decoy application.

We used a dataset with 27,000 actual TCP flows consisting of the traffic of six different users for a collective duration of 28 days. We use the TCP flows generated by the WGAN to carry the actual network traffic as the payload of the generated packets as a tunnel. In this way, an adversary who aims to detect the originating

applications would classify the WGAN-generated flows as being generated by the decoy applications.

We compared the distribution of generated flows to the distribution of actual network traffic of six popular applications, and we showed that the distribution of the generated flows closely follows that of the actual network traffic. We tested the efficacy of our proposed GAN tunnel traffic by evaluating the classification performance of RF and XGBoost, which are qualified examples of effective adversarial ITCs. The results show that these classifiers detect the flows as being generated by the decoy applications with an average accuracy of 0.99 when these classifiers and the WGAN are trained on the same dataset, and 0.97 when these classifiers and WGAN are trained on two separate datasets. Our tests show that the GAN tunnel effectively masks the statistical properties of the traffic generated by user applications and that it can make it appear as generated by a selectable decoy application.

CHAPTER 5

CONCLUSION

In this study, we studied Internet traffic classifiers (ITCs), employing machine learning algorithms for traffic classification. We showed that ITCs are capable of identifying user online activities with high accuracies up to 99%. We discussed that ITCs might be used for adversarial purposes as well, such as website fingerprinting attacks in which an intruder analyzes encrypted traffic of a user to infer a user's online activities from statistical traffic patterns. Protecting users' online privacy against traffic classification attacks is the primary motivation of this work.

We show the effectiveness of ITCs by comparing the performance of 11 ITCs employing state-of-art machine learning algorithms. We proposed three traffic anonymization methods to counter the adversarial use of ITCs. These methods are Equalized packet length, Equalized packet count, and Equalized inter-arrival times of TCP packets. These methods reduce the detectability of network traffic patterns from adversaries by equalizing the statistical properties of the network traffic. We showed that these methods are capable of reducing the precision of ITCs detection up to 90%. Among these methods, Equalized packet length is the most effective anonymization method. We also showed that XGBoost and Bernoulli NB are the most resilient classifiers against these countermeasures, because their reliance on packet length, packet count, and inter-arrival times is less than the other tested ML algorithms.

To improve network traffic anonymization against adversarial ITCs, we proposed a GAN tunnel, which is based on a generative adversarial network (GAN). GAN tunnel generates synthetic traffic patterns imitating the real network traffic generated by actual applications. It then encapsulates the actual network packets into the generated traffic flows to hide them from adversaries. We showed that adversarial ITCs could not detect the encapsulated traffic source application by using the GAN

tunnel. We tested GAN tunnel traffic against ITCs employing Random Forest and XGBoost. These ITCs were entirely incapable of detecting the actual source application.

To extend our work in the future, we will study differential privacy applicability in network traffic analysis. Differential privacy ensures that the gathered data from users does not lead to the identification of individuals while allowing data analysis on the data [110]. In the case of traffic analysis, this means that we can analyze gathered information from network traffic to train our classifiers for purposes such as intrusion and anomaly detection. However, these collected data should be saved in a deferentially private database that does not let the classifier identify the individuals who generated the network traffic.

Another privacy-preserving technique for the gathered data is called k-anonymity privacy requirement [111]. k-anonymity requires that each equivalent class of data in a database contain at least k records. In the case of network traffic analysis, let us consider a dataset containing labeled data of three different users. This dataset's features include IP addresses, average packet lengths, and the name of the traffic-generating application. To preserve an adversary from identifying an individual's IP address based on a prior knowledge about applications in use by that adversary and the average packet length of those application, the dataset is anonymized so that the average packet-length column includes ordinal information instead of cardinal numbers. Additionally, the IP addresses should be truncated to have first and second octets instead of all four octets. This way, the IP addresses cannot be directly traced back to a user's device. By applying these changes and grouping each k rows of data to include similar anonymized IP addresses and ordinal average packet lengths, we get a dataset with k-anonymity.

REFERENCES

[1] S. Fathi-Kazerooni, Y. Kaymak, and R. Rojas-Cessa, "Tracking user application activity by using machine learning techniques on network traffic," in *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Feb 2019, pp. 405–410.

[2] S. Fathi-Kazerooni and Y. Kaymak and R. Rojas-Cessa, "Identification of user application by an external eavesdropper using machine learning analysis on network traffic," in *IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2019, pp. 1–6.

[3] P. Perera, Y.-C. Tian, C. Fidge, and W. Kelly, "A comparison of supervised machine learning algorithms for classification of communications network traffic," in *International Conference on Neural Information Processing*. Springer, 2017, pp. 445–454.

[4] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 5–16, 2006.

[5] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[6] A. K. Sharma and P. S. Parihar, "An effective dos prevention system to analysis and prediction of network traffic using support vector machine learning," *International Journal of Application or Innovation in Engineering & Management*, vol. 2, no. 7, pp. 249–256, 2013.

[7] M. Lotfollahi, R. Shirali, M. J. Siavoshani, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *arXiv preprint arXiv:1709.02656*, 2017.

[8] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th Annual Workshop on Privacy in the Electronic Society*. ACM, 2011, pp. 103–114.

[9] F. Zhang, W. He, X. Liu, and P. G. Bridges, "Inferring users' online activities through traffic analysis," in *Proceedings of the fourth conference on Wireless network security*. ACM, 2011, pp. 59–70.

[10] B. A. Forouzan, *TCP/IP protocol suite*. New York, NY: McGraw-Hill, Inc., 2002.

[11] J. Li, L. Zhou, H. Li, L. Yan, and H. Zhu, "Dynamic traffic feature camouflaging via generative adversarial networks," in *IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 268–276.

[12] Z. Cao, G. Xiong, Y. Zhao, Z. Li, and L. Guo, "A survey on encrypted traffic classification," in *International Conference on Applications and Techniques in Information Security*. Springer, 2014, pp. 73–81.

[13] F. Yu, Z. Chen, Y. Diao, T. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in *Proceedings of the ACM/IEEE symposium on Architecture for networking and communications systems*. ACM, 2006, pp. 93–102.

[14] Service name and transport protocol port number registry. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (last accessed May 1, 2018)

[15] H. A. H. Ibrahim, O. R. A. Al Zuobi, M. A. Al-Namari, G. MohamedAli, and A. A. A. Abdalla, "Internet traffic classification using machine learning approach: Datasets validation issues," in *Conference of Basic Sciences and Engineering Studies (SGCAC)*. IEEE, 2016, pp. 158–166.

[16] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *The IEEE Conference on Local Computer Networks. 30th Anniversary.* IEEE, 2005, pp. 250–257.

[17] 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.

[18] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: data collection and traffic classification," in *IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 2016, pp. 1–5.

[19] G. Sun, T. Chen, Y. Su, and C. Li, "Internet traffic classification based on incremental support vector machines," *Mobile Networks and Applications*, pp. 1–8, 2018.

[20] Z. Fan and R. Liu, "Investigation of machine learning based network traffic classification," in *International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, 2017, pp. 1–6.

[21] W. Li and A. W. Moore, "A machine learning approach for efficient traffic classification," in *15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. MASCOTS'07.* IEEE, 2007, pp. 310–317.

[22] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 50–60.

[23] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *International Workshop on Passive and Active Network Measurement*. Springer, 2004, pp. 205–214.

[24] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.

[25] J. Wan, L. Wu, Y. Xia, J. Hu, Z. Xia, R. Zhang, and M. Wang, "Classification method of encrypted traffic based on deep neural network," in *International Conference of Pioneering Computer Scientists, Engineers and Educators*. Springer, 2019, pp. 528–544.

[26] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: smartphone fingerprinting via application behaviour," in *Proceedings of the sixth conference on Security and privacy in wireless and mobile networks*. ACM, 2013, pp. 7–12.

[27] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 512–521.

[28] B. Yamansavascilar, M. A. Guvensan, A. G. Yavuz, and M. E. Karsligil, "Application identification via network traffic classification," in *International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2017, pp. 843–848.

[29] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, G. Wei, and L. T. Yang, "Internet traffic classification using constrained clustering," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 11, pp. 2932–2943, 2013.

[30] J. Erman, A. Mahanti, and M. Arlitt, "Qrp05-4: Internet traffic identification using machine learning," in *IEEE Global Telecommunications Conference. GLOBECOM'06.* IEEE, 2006, pp. 1–6.

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

[32] S. Fathi-Kazerooni and R. Rojas-Cessa, "GAN Tunnel: Network Traffic Steganography by Using GANs to Counter Internet Traffic Classifiers," *IEEE Access*, vol. 8, pp. 125 345–125 359, 2020.

[33] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2017.

[34] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[35] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Mimetic: Mobile encrypted traffic classification using multimodal deep learning," *Computer Networks*, vol. 165, p. 106944, 2019.

[36] Wireshark. [Online]. Available: https://www.wireshark.org (last accessed May 1, 2018)

[37] Wireshark paint. [Online]. Available: https://www.digitaloperatives.com/project/paint/ (last accessed May 1, 2018)

[38] Event tracing for windows. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows–etw- (last accessed May 1, 2018)

[39] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. New York, NY: Springer series in statistics, 2001, vol. 1, no. 10.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[41] S. Boyd and L. Vandenberghe, *Convex optimization*. New York, NY: Cambridge University Press, 2004.

[42] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. Cambridge, MA: MIT press, 2009.

[44] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2019.

[45] S. Kay, *Intuitive probability and random processes using MATLAB®*. New York, NY: Springer Science & Business Media, 2006.

[46] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[47] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes?" in *CEAS*, vol. 17. Mountain View, CA, 2006, pp. 28–69.

[48] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. Hoboken, NJ: John Wiley & Sons, 2012, vol. 821.

[49] G. E. Box, G. M. Jenkins, and G. C. Reinsel, *Time series analysis: forecasting and control*. Hoboken, NJ:John Wiley & Sons, 2011, vol. 734.

[50] S. Fathi-Kazerooni, R. Rojas-Cessa, Z. Dong, and V. Umpaichitra, "Time Series Analysis and Correlation of Subway Turnstile Usage and COVID-19 Prevalence in New York City (In Press)," *Infectious Disease Modelling*, pp. 1–11, 2020.

[51] F. Flandoli. (2020) ARIMA models. [Online]. Available: http://users.dma.unipi.it/ flandoli/AUTCap4.pdf

[52] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[54] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, 2017, pp. 214–223.

[55] B. Bailey and M. J. Telgarsky, "Size-noise tradeoffs in generative networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6489–6499.

[56] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[57] E. Heim, "Constrained generative adversarial networks for interactive image generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 753–10 761.

[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, MA: MIT press, 2016.

[59] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 5767–5777. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf

[60] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," 2017.

[61] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[62] I. Chakravati, R. Laha, and J. Roy, *Handbook of Methods of Applied Statistics, Volume I.* New York: John Wiley and Sons, 1967.

[63] P.-C. Lin, B. Wu, and J. Watada, "Kolmogorov-smirnov two sample test with continuous fuzzy data," in *Integrated Uncertainty Management and Applications*. Springer, 2010, pp. 175–186.

[64] A. Sun and E.-P. Lim, "Hierarchical text classification and evaluation," in *Proceedings of IEEE International Conference on Data Mining. ICDM.* IEEE, 2001, pp. 521–528.

[65] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[66] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[67] Most popular internet browser versions 2018. [Online]. Available: https://www.statista.com/statistics/268299/most-popular-internet-browsers (last accessed May 1, 2018)

[68] Top U.S. mobile social apps by users 2018. [Online]. Available: https://www.statista.com/statistics/248074/most-popular-us-social-networking-apps-ranked-by-audience (last accessed May 1, 2018)

[69] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis." in *NDSS*, vol. 9. Citeseer, 2009.

[70] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th conference on Computer and communications security.* ACM, 2006, pp. 255–263.

[71] Z. Li, R. Yuan, and X. Guan, "Accurate classification of the internet traffic based on the svm method," in *IEEE International Conference on Communications.* IEEE, 2007, pp. 1373–1378.

[72] A. Dainotti, F. Gargiulo, L. I. Kuncheva, A. Pescapè, and C. Sansone, "Identification of traffic flows hiding behind tcp port 80," in *IEEE International Conference on Communications.* IEEE, 2010, pp. 1–6.

[73] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1194–1213, 2007.

[74] T. Bakhshi and B. Ghita, "On internet traffic classification: A two-phased machine learning approach," *Journal of Computer Networks and Communications*, vol. 2016, 2016.

[75] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2012, pp. 332–346.

[76] X. Fu, B. Graham, R. Bettati, and W. Zhao, "Active traffic analysis attacks and countermeasures," in *International Conference on Computer Networks and Mobile Computing, 2003. ICCNMC 2003.* IEEE, 2003, pp. 31–39.

[77] X. Fu, B. Graham, R. Bettati, W. Zhao, and D. Xuan, "Analytical and empirical analysis of countermeasures to traffic analysis attacks," in *2003 International Conference on Parallel Processing, 2003. Proceedings.* IEEE, 2003, pp. 483–492.

[78] L. Chaddad, A. Chehab, I. H. Elhajj, and A. Kayssi, "Mobile traffic anonymization through probabilistic distribution," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2019, pp. 242–248.

[79] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[80] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[81] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on systems, Man, and Cybernetics*, no. 6, pp. 448–452, 1976.

[82] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

[83] dpkt python library. [Online]. Available: https://github.com/kbandla/dpkt (last accessed Aug 1, 2020)

[84] T. Ylonen and C. Lonvick, "The secure shell (ssh) protocol architecture," 2006.

[85] M. Shanker, M. Y. Hu, and M. S. Hung, "Effect of data standardization on neural network training," *Omega*, vol. 24, no. 4, pp. 385–397, 1996.

[86] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

[87] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[88] P. Baldi and P. J. Sadowski, "Understanding dropout," in *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013, pp. 2814–2822. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf

[89] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 2146–2153.

[90] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[91] F. Chollet *et al.* (2015) Keras. https://keras.io (last accessed May 1, 2018).

[92] K. P. Balanda and H. MacGillivray, "Kurtosis: a critical review," *The American Statistician*, vol. 42, no. 2, pp. 111–119, 1988.

[93] J. J. Jenq and W. Li, "Feedforward backpropagation artificial neural networks on reconfigurable meshes," *Future Generation Computer Systems*, vol. 14, no. 5-6, pp. 313–319, 1998.

[94] K.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004.

[95] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 4, pp. 325–327, 1976.

[96] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*. Sebastopol, CA: O'Reilly Media, Inc., 2013.

[97] J. Charlier, A. Singh, G. Ormazabal, R. State, and H. Schulzrinne, "Syngan: Towards generating synthetic network attacks using gans," *arXiv preprint arXiv:1908.09899*, 2019.

[98] L. Vu, C. T. Bui, and Q. U. Nguyen, "A deep learning based method for handling imbalanced problem in network traffic classification," in *Proceedings of the Eighth International Symposium on Information and Communication Technology*, 2017, pp. 333–339.

[99] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers & Security*, vol. 82, pp. 156–172, 2019.

[100] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," *arXiv preprint arXiv:1702.05983*, 2017.

[101] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative adversarial networks for attack generation against intrusion detection," *arXiv preprint arXiv:1809.02077*, 2018.

[102] M. Rigaki and S. Garcia, "Bringing a gan to a knife-fight: Adapting malware communication to avoid detection," in *IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 70–75.

[103] A. Cheng, "PAC-GAN: Packet generation of network traffic using generative adversarial networks," in *IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2019, pp. 0728–0734.

[104] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri, and M. Conti, "On defending against label flipping attacks on malware detection systems," *Neural Computing and Applications*, pp. 1–20, 2020.

[105] S. Garcia, "Modelling the network behaviour of malware to block malicious patterns. the stratosphere project: a behavioural ips," *Virus Bulletin*, pp. 1–8, 2015.

[106] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic Regression*. New York, NY: Springer, 2002.

[107] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine learning*, vol. 36, no. 1-2, pp. 105–139, 1999.

[108] R. Real and J. M. Vargas, "The probabilistic basis of jaccard's index of similarity," *Systematic biology*, vol. 45, no. 3, pp. 380–385, 1996.

[109] B. Ramsundar and R. B. Zadeh, *TensorFlow for deep learning: from linear regression to reinforcement learning*. Sebastopol, CA: O'Reilly Media, Inc., 2018.

[110] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy." *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[111] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 2007, pp. 106–115.