Received 14 June 2023, accepted 30 June 2023, date of publication 10 July 2023, date of current version 14 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3293526

RESEARCH ARTICLE

Darknet Traffic Analysis: Investigating the Impact of Modified Tor Traffic on Onion Service Traffic Classification

ISHAN KARUNANAYAKE[1,3], (Graduate Student Member, IEEE), NADEEM AHMED[1,3], ROBERT MALANEY[1,3], (Senior Member, IEEE), RAFIQUL ISLAM[2,3], AND SANJAY K. JHA[1,3], (Senior Member, IEEE)

[1]Institute for Cybersecurity (IFCYBER), University of New South Wales (UNSW), Sydney, NSW 2052, Australia
[2]School of Computing and Mathematics, Charles Sturt University, Albury, NSW 2640, Australia
[3]Cyber Security Cooperative Research Centre, Joondalup, WA 6027, Australia

Corresponding author: Ishan Karunanayake (ishan.karunanayake@unsw.edu.au)

This work has been supported by the Cyber Security Research Centre Limited (CSCRC) whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme.

ABSTRACT Classifying network traffic is important for traffic shaping and monitoring. In the last two decades, with the emergence of privacy concerns, the importance of privacy-preserving technologies has risen. The Tor network, which provides anonymity to its users and supports anonymous services known as *Onion Services*, is a popular way to achieve online anonymity. However, this anonymity (especially with Onion Services) is frequently misused, encouraging governments and law enforcement agencies to de-anonymise them. Therefore, in this paper, we try to identify the classifiability of Onion Service traffic, focusing on three main contributions. First, we try to identify Onion Service traffic from other Tor traffic. The techniques we have used can identify Onion Service traffic with >99% accuracy. However, there are several modifications that can be done to the Tor traffic to obfuscate its information leakage. In our second contribution, we evaluate how our techniques perform when such modifications have been done to the Tor traffic. Our experimental results show that these conditions make the Onion Service traffic less distinguishable (in some cases, the accuracy drops by more than 15%.) In our final contribution, we identify the most influential feature combinations for our classification problem and evaluate their impact.

INDEX TERMS Traffic classification, machine learning, onion services, tor, anonymity, feature selection.

I. INTRODUCTION

Tor [1] is an anonymity network that hides the identity of its users by routing the traffic through multiple intermediary nodes. Tor also supports the provision of anonymous services known as Onion Services (also known as hidden services) with *.onion* as the top-level domain name. Tor's ability to act as a censorship circumvention tool has encouraged security experts, network defenders, and law enforcement agencies to identify Tor traffic from other encrypted and non-encrypted traffic [2], [3]. For example, [3], [4] tried to classify Tor traffic from non-Tor Traffic, [2], [5] tried to classify the application types in Tor traffic, and [6] tried to classify Tor traffic from other anonymity network traffic such as I2P traffic and Web-mix Traffic. However, in this work, we intend to explore the distinguishability of Onion Service traffic from standard Tor traffic using traffic analysis. We formulate three research questions to act as a foundation for our work.

First, we try to answer the question, ***RQ1: Is it possible to classify Onion Service Traffic from other standard Tor traffic?*** A standard Tor circuit that is created to visit a web service on the Internet via Tor consists of three Tor nodes. An Onion Service circuit, which is the only way to access an Onion Service, consists of six Tor nodes. As the traffic in both these circuits (standard Tor and Onion Service) is encrypted, we assume that we can use the information leaked from the

The associate editor coordinating the review of this manuscript and approving it for publication was Nazar Zaki.

metadata (e.g. direction, timestamps, packet size) to identify unique patterns that can distinguish them. Onion Services have been used to host illegal websites, and more recently, they have been used as Command and Control (C&C) servers for botnets [7], [8]. Therefore, from the perspective of governments and law enforcement agencies, they want to track and shut down such services and regulate the Onion Service traffic [9]. Even businesses might find it useful to restrict access to such websites in order to protect their systems from potential bad actors (e.g. hackers) and attacks. As a result, having techniques for identifying Onion Service traffic can be useful for two main reasons; 1. Such techniques can act as a stepping stones for fingerprinting of Onion Services. 2. They can be useful to restrict Onion Service traffic in sensitive and confidential systems.

Second, we try to investigate the same problem under different settings. Specifically, we try to investigate ***RQ2: How do our results for RQ1 hold when we use modified Tor traffic?*** There are certain techniques that can be implemented in Tor to change its traffic patterns. Introducing padding [10], using dummy bursts and delays [11], and splitting the traffic [12] are a few examples of such techniques. These techniques[1] have been developed with the intention of obfuscating the information leakage of Tor traffic. The main importance of answering RQ2 is that we can confirm whether our findings from RQ1 will hold true as and when such modifications are introduced to the Tor traffic. If we are able to still distinguish Onion Service traffic, it is an indication that these modifications are not effective in masking Onion Service traffic, if they are realised in the future. If the modifications do affect the Onion Service classifiability, it opens up questions about the validity of prior works, such as [3] and [6] in a setting with those modifications implemented. As outcomes of RQ2 can open up further research avenues on Tor traffic classification, we argue RQ2 is worth evaluating.

In order to investigate RQ1 and RQ2, we employ passive network analysis, in which we utilise traces of network traffic captured at points between the client and the Tor entry node. We first extract fifty features from each traffic trace, which creates a unique fingerprint of that trace. A traffic trace refers to a set of consecutive packets transmitted between the client and the entry node in a given duration. We use three machine learning classifiers that have shown promising results in network traffic classification in the past and evaluate how they work in our scenarios.

As our third research question, we investigate ***RQ3: What features impact Onion Service traffic classification most, and what level of performance do they provide?*** We considered two factors when crafting the fifty features used in this work. (i) We mentioned that the standard Tor traffic passes through three Tor nodes, while the Onion Service traffic passes through six. Intuitively, this difference should lead to major differences in latency. Therefore, we focused on features that are focused on timing statistics. (ii) Also, we use features that have a proven track record of working well in revealing patterns in network traffic [13]. However, we use three feature selection techniques to infer which features have a better relationship with the traffic types used in our work and conduct experiments to evaluate the classifier performance with different feature combinations.

[1]These techniques are commonly identified as Website Fingerprinting defences. Website Fingerprinting refers to a passive de-anonymisation attack executed on Tor users, where an adversary tries to identify a user's online activity. More information is provided in Section II.

Overall, we make the following specific contributions in this work.

1) First, we try to classify standard Tor traffic and Onion Service traffic. We evaluate the applicability of three supervised machine learning algorithms, namely K-Nearest Neighbor, Random Forest, and Support Vector Machines, for our evaluations. We also extract fifty different features, which are given as input to the machine learning classifiers. Our results show that most of these algorithms can identify Onion Service traffic from other Tor traffic with a high degree of accuracy (given that we use effective features and sufficient samples). We find Random Forests can predict the results more than 100 times faster than other techniques, making it an ideal candidate for real-world online implementations.
2) Second, we further evaluate the identifiability of Onion Service traffic, when certain modifications (e.g. padding, traffic splitting) are introduced to Tor traffic. We use Tor traffic generated with two techniques, WTFPAD [10], and TrafficSliver [12] in our experiments. Our experiments show that the accuracy values we obtained for RQ1 drop considerably (in some cases, even by more than 15%) when these modifications are introduced. This provides a strong indication that such modifications can affect the results obtained in prior Tor traffic classification works (listed in Table 1).
3) Then, we use different feature selection metrics (Information Gain, Pearson Correlation, and Fisher Score) to select different feature combinations that have the most effect on the classification and investigate the performance of the classifiers. Our results provide insights into the importance of different features and suggest that the higher performance of the classifiers is, in fact, highly dependent on the selection of features.

The rest of the paper is organised as follows. In Section II, we provide background and related work for Tor traffic classification. We describe our dataset and features in Section III. In Section IV, we explain the experimentation process with the results. In Section V, we discuss several insights we obtain from our work along with potential future directions. We conclude in Section VI.

II. BACKGROUND AND RELATED WORK

In this section, we provide background information on the Tor network and Onion Services, which is useful in understanding our work and present the related work on Tor Traffic Classification.

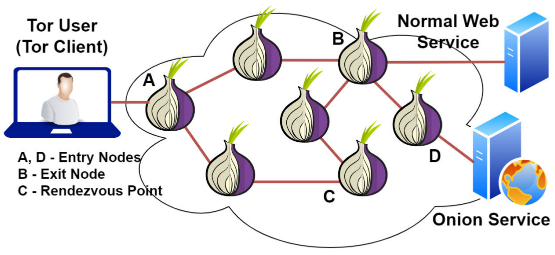


FIGURE 1. Connections to a normal web service and an Onion Service via Tor.

A. TOR NETWORK AND ONION SERVICES

The Tor network is comprised of multiple relays as shown in Figure 1. When a user connects to a destination service, their traffic is encrypted and routed through a selected set of intermediary relays. In the case where a user connects to a normal web service through Tor, the Tor circuit generally consists of three relays known as the entry, middle, and exit. Although this setup adds some latency to the communication, it provides strong anonymity to the user. An entity that has the ability to monitor the traffic at any point of the Tor circuit is unable to link a user with their destination (e.g. finding their IP addresses). However, in this scenario, the web service is not anonymous. Its IP address or the domain name is public knowledge. Tor has introduced Onion Services to overcome this problem. An Onion Service is simply a normal web service that can only be accessed via the Tor network. A user accessing an Onion Service does not know the actual IP address (hence the location) of the Onion Service. When a user connects to an Onion Service, the circuit generally consists of six relays and is created as follows. The Onion Service first selects a few random relays from the Tor network as its *Introduction Points*. Then it advertises a service descriptor containing the addresses of the introduction points and the Onion Service's public key in another Tor node called the Hidden Service Directory (HSDir). Next, the Onion Service operator advertises the .onion address of the Onion Service to potential users, normally via other Onion Services, blogs, and social media. A user requires a Tor client (a small piece of software that can manage Tor-related operations) to search for this Onion Service. The Tor client retrieves the service descriptor of the Onion Service from the relevant HSDir. The Tor client then selects a random Tor relay known as the Rendezvous Point (RP), establishes a circuit to it and sends a message to the Onion Service via the introduction points. This message contains the address of the RP and a one-time cookie. Finally, the Onion Service initiates a Tor connection to the RP and completes the circuit [1].

B. MODIFIED TOR TRAFFIC

As mentioned previously, Website Fingerprinting is a passive de-anonymisation attack that can be executed with minimal resources. In a conventional Website Fingerprinting attack, the attacker trains a machine learning classifier to identify the websites visited by a user. Tor traffic collected between the Tor client and the entry node is used to extract features for the classifier. Once the attacker has an effective model, they can use it to determine the websites a Tor user is visiting by intercepting the traffic and feeding its fingerprint to the model. The success of this attack depends on the information leakage of Tor traffic (usually inferred using metadata as the payload is encrypted). Therefore, several defences such as WTFPAD [10], and TrafficSliver [12] have been proposed to obfuscate the information leakage in Tor traffic.

- **WTFPAD:** This defence uses adaptive padding and tries to conceal traffic bursts and other traffic features. Here adaptive padding refers to padding when the channel is not being used. That way, there will be fake traffic during channel idle times, affecting the formation of unique patterns.
- **TrafficSliver:** This technique uses a traffic splitting and multipathing strategy. It means that the network traffic is split and sent over multiple entry nodes. As this technique does not add any additional packets or delays, it is more efficient than other defences.

In this paper, we aim to understand how these modifications affect the classifiability of Tor Traffic.

C. MACHINE LEARNING

Machine learning is a major sub-area under artificial intelligence and is widely adopted for applications such as network traffic classification and malware detection. In general, when using a machine learning system for encrypted traffic classification tasks, first, it is necessary to collect traffic traces under relevant (and realistic) conditions. Then, important features such as packet sizes, inter-arrival times, and the number of packets are extracted from these traffic traces. After obtaining this data, it is important to process and clean the data before evaluation through machine learning algorithms. Checking for missing values, removing duplicates, and handling categorical values are some steps used in the data pre-processing stage. Once there is a processed dataset, it is split into two main parts; training and testing. The next step is to use a suitable machine learning classifier on the training dataset and build a model. Finally, the model is evaluated on the testing dataset using metrics such as *accuracy*, *precision*, and *recall* (these metrics will be detailed in Section IV). Moreover, it can be useful to investigate the performance of a model in terms of training and prediction times. There are two main types of problems handled by machine learning; regression and classification. In this work, we focus on the classification capabilities of machine learning and use three traditional machine learning algorithms in our experiments. We use the term traditional to describe machine learning algorithms that do not use Artificial Neural Networks (ANNs). While these traditional machine learning algorithms need more human intervention, especially with preparing data and tuning hyper-parameters compared to ANNs, they have their advantages. For example, traditional machine learning algorithms can work well with fewer data than ANNs in general, and the results are much more interpretable.

- **K-Nearest Neighbour (KNN):** This algorithm selects the K (an integer) number of nearest data points for a new data point and classifies the new data point into the most common category of the K nearest points.
- **Random Forest (RF):** RF builds multiple decision trees by using randomly selected features and combining them together. A decision tree refers to a model that learns from training data while developing decision rules for different outcomes. During the prediction phase, these rules help to determine the target outcome for a given input. Having multiple decision trees enable RF to provide more accurate predictions.
- **Support Vector Machines (SVM):** SVM tries to find the best hyperplane that can separate multiple classes. It tries to find the hyperplane, which divides the classes, while having the largest distances between the hyperplane and the nearest elements of each class.

D. RELATED WORK

There are prior works that tried to address other questions related to classifying Tor traffic. For example, classifying Tor traffic from non-Tor traffic, classifying Tor traffic from other anonymity networks' traffic, etc. Different works in the literature have employed different machine learning-based classification methods with different sets of features. We will describe these works as related work in this section.

Bai et al. [14] attempted to identify Tor traffic and Web-mix traffic [15] by using a stepwise matching technique. They extracted different types of fingerprints, including information in the packet header, specific strings in the packets, and some statistical features such as packet length and frequency. Their approach could identify Tor traffic with an *accuracy* of 95.98%.

AlSabah et al. [2] tried to classify Tor traffic into the type of application that is being used. They considered *interactive web browsing* and *bulk downloading* (mainly BitTorrent and streaming applications) as the application types in their work. They identified that bulk downloading takes up a large bandwidth while contributing to a very small percentage of the total connections. The aim of the authors in [2] was to provide different Quality of Service (QoS) to different traffic classes. They used features such as cell Inter-Arrival Times (IAT), circuit lifetime, amount of data sent upstream and downstream, and classification algorithms such as Naive Bayes, Bayesian Networks, and Decision Trees. The experiments of [2] show an *accuracy* of over 95% in classifying the application type of Tor circuits on a live Tor network.

He et al. [5] also tried to determine the application type the encrypted Tor Traffic contained. In contrast to AlSabah et al.'s work, He et al. considered the following application types; P2P, Web, File Transfer Protocol (FTP), and Instant Messaging (IM). They used Profile Hidden Markov Models (PHMM) as their classifier and flow-based features such as burst volumes and direction of packets as features. They obtained *accuracy* figures up to 92% in their experiments.

Lashkari et al. [3] evaluated how Tor traffic can be distinguished from non-Tor traffic by only using time-based features. They extracted 23 time-based features from the traffic they captured and used machine learning algorithms such as K-Nearest Neighbor, Decision Trees and Random Forests. Their techniques could classify Tor traffic from non-Tor traffic with a *precision* and *recall* of more than 95%. In addition, they tried to classify Tor traffic into different applications, including VoIP, Video-streaming, audio-streaming, browsing, chat, FTP, and P2P and achieved *precision* and *recall* values of around 80%.

Deep Learning (DL) is a subset of machine learning, which is widely used in applications such as image classification and speech recognition. Generally, algorithms that consist ANNs fall under this category. Kim and Anpalagan [4] applied Convolutional Neural Networks (CNN) to classify Tor traffic from non-Tor traffic. They used hexadecimal raw packets with a CNN and obtained an overall *accuracy* of 99.3% in identifying different application types. They used the same dataset used by Lashkari et al. [3] and showed that their method outperformed the techniques in [3].

Although Tor is very popular as an anonymity network, it is not the only anonymity system that is currently in use. Montieri et al. [6] carried out experiments to classify traffic from different anonymity systems, including Tor, I2P [16], and JonDonym (formerly known as Web- Mix) [15]. They used four sets of features, including flow-based statistics (e.g. flow direction, duration, inter-arrival time statistics), histogram representations of packet lengths and inter-arrival times, and sequence of packets. Bayesian classifiers and tree-based classifiers were used in their work. Their results show an *accuracy* of 99.87% for classifying traffic belonging to different networks and 73.99% for determining the application type by using flow-based features.

III. DATASET AND FEATURE SELECTION

In this section, we present the details of the datasets we used in our study. Also, we provide information on the features and feature selection metrics we used.

A. DATASETS

We use two publicly available datasets, which we refer to in this paper as TOR (representing standard Tor traffic) and OS (representing Onion Service traffic), in our experiments. The TOR [17] dataset consists of 95000 traffic traces collected by accessing 95 websites (1000 traces per site) over the Tor Network. The OS [18] dataset contains 41503 traces from 539 Onion Services (77 traces per site). Both these datasets have been collected using a similar approach from the real Tor network and do not contain any modifications we need for our experiments in RQ2. We also created four new simulated datasets using the techniques proposed in [10] and [12]. We refer to these as 'WTFPAD-TOR', 'WTFPAD-OS', 'TrafficSliver-TOR', and 'TrafficSliver-OS'. We refer

TABLE 1. Summary of related work.

Publication	Objective	Features	Classifier
Bai *et al.* [14]	Identify Tor and WebMix traffic	Packet length, frequency of packets	Rule Based methods
AlSabah *et al.* [2]	Classify the application type (streaming, bitTorrent, browsing) in Tor traffic	Circuit lifetime, cell inter-arrival time, recent no. of cells	Naïve Bayes, Baysian Networks, Decision Trees
He *et al.* [5]	Classify application type (Web, P2P, etc) in Tor traffic	Burst volumes and direction of packets	Profile Hidden Markov Models
Lashkari *et al.* [3]	Classify Tor traffic from non-Tor Traffic and identify the application Type	23 Time based features	ZeroR, C4.5, KNN, Random Forest
Kim et al. [4]	Classify Tor traffic from non-Tor Traffic and identify the application Type	Features based on hexadecimal raw packet header	Convolutional Neural Networks
Montieri et al. [6]	Classify different anonymity network traffic (Tor, I2P, JonDonym)	flow-based statistics, histogram representations of packet lengths, inter-arrival times, and sequence of packets	Bayesian and tree-based classifiers
Our paper	Onion Service traffic classification, evaluating the impact of modified Tor traffic, and the feature importance on classification	50 statistical features	KNN, Random Forest, SVM

readers to the original papers for more information regarding the data collection process [17], [18] and the simulation of traffic modifications [10], [12].

B. FEATURE EXTRACTION

Each packet's timestamp and direction (incoming or outgoing) are included in the datasets we have available. From that information, we extract fifty statistical features from each traffic trace to be used in our experiments. We selected many features which involve timing statistics, and used insights and features identified from prior works, [13]. We then carry out a set of experiments to identify which of those features works best for our research question (see next subsection). The details of these features are provided below, where we assign a number (1-50) to each feature so as to easily identify them in Figure 2.

1) THIRTEEN FEATURES BASED ON INTER-ARRIVAL TIMES (IATS)

IAT is the time difference between two successive packets to arrive at a particular point. We extract the maximum (**1**), mean (**2**), Standard Deviation (SD) (**3**), and the 75th percentile (**4**) of the IATs for all incoming packets (from the entry node to the Tor client), max (**5**), mean (**6**), SD (**7**), and the 75th percentile (**8**) of IATs for outgoing packets (from the Tor client to the entry node), and max (**9**), mean (**10**), SD (**11**), and the 75th percentile (**12**) for all packets. Finally, we calculate the sum of all the IAT-related features (**13**).

2) THIRTEEN FEATURES BASED ON THE FLOW DURATION

Here, we calculate features related to the duration of the flow with respect to the starting time of that flow. For example, let us take the 25th percentile duration for incoming packets. To calculate this feature, we first separate all the incoming packets from that flow. Then we find the 25th percentile packet in the incoming packet sequence, and calculate the time that packet was received since the start of the flow (**14**). Similarly, we calculate the 50th percentile (**15**), 75th percentile (**16**), and the total duration (**17**) for incoming packets. Then, we calculate the same features for the outgoing packets (25th (**18**), 50th (**19**), 75th (**20**), total (**21**)) and finally for all packets (25th (**22**), 50th (**23**), 75th (**24**), total (**25**)). Finally, we calculate the sum of those duration-related features (**26**).

3) EIGHT FEATURES BASED ON THE NUMBER OF PACKETS

The number of incoming packets (**27**), outgoing packets (**28**) and the total number of packets (**29**) in a flow were extracted from the dataset. In addition, we calculated the number of incoming (**30**) and outgoing packets (**31**) among the first 30 packets and last 30 packets (incoming (**32**), outgoing (**33**)). Finally, the sum of the values of these latter seven features (**34**) was calculated.

4) FIVE FEATURES BASED ON THE PACKET CONCENTRATION

When calculating these features, we first divided a trace into segments of 20 successive packets. Then we calculated the number of outgoing packets in each of the segments and recorded the values sequentially. Finally, we calculated the mean (**35**), min (**36**), max (**37**), SD (**38**), and median (**39**) of those values.

5) FIVE FEATURES BASED ON THE PACKET FREQUENCY

Defining the packet frequency as the average number of packets transmitted per second, we calculated the mean (**40**), max (**41**), min (**42**), SD (**43**), and median (**44**) of the packet frequencies.

6) FOUR FEATURES BASED ON THE PACKET ORDER

These are a set of features where we first separate the incoming and outgoing packets in a flow. Then we number the packets in each group from zero and create two separate lists. For example, if there are 5 incoming packets and 3 outgoing packets in a flow, we number the incoming packets as 0,1,2,3 and 4 and the outgoing packets as 0,1,2. Finally, we calculate the mean **(45)** and the SD **(46)** of the incoming packet list and the outgoing packet list (mean **(47)**, SD **(48)**).

7) TWO FEATURES BASED ON THE PACKET PERCENTAGE

These last two features simply provides the number of incoming **(49)** and outgoing packets **(50)** as a fraction of the total number of packets in the flow.

C. FEATURE SELECTION

We use three metrics that help quantify the importance of a feature: Information gain, Pearson correlation, and Fisher Score. More details about these metrics and how we use them are described below.

- **Information gain (IG)** evaluates the importance of a particular feature by measuring its information gain with respect to the class [19]. It is calculated as the difference between the entropy of the class before and after splitting the dataset on the feature as shown by Equation 1.

$$IG\ (Class,\ Feature) = H(Class)\ \text{-}\ H(Class \mid Feature), \quad (1)$$

  where $H(Class)$ is the entropy of the class and $H(Class|Feature)$ is the conditional entropy of the class given the feature. In general, the entropy $H(X)$ of a class X is calculated as shown in Equation 2.

$$H(X) = -\sum_{i=1}^{N} p_i \log_2 p_i, \quad (2)$$

  where N is the number of classes and p_i is the proportion of instances of the i^{th} class.
  In a machine learning context, entropy is used to measure the degree of impurity (degree of uncertainty) in a dataset with respect to the class labels. As suggested by Equations 1 and 2, a high information gain indicates that the entropy of the class labels is significantly reduced after the splitting of the dataset compared to the original dataset's entropy. It shows that the split has reduced the degree of uncertainty and suggests that the feature is important in distinguishing the different classes of the dataset.
- **Pearson's correlation** shows how strongly two variables are correlated linearly and is defined as follows.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \quad (3)$$

  where r is the Pearson's correlation coefficient, x_i and y_i are the i^{th} data points of the variables x and y, $\bar{x}$ and $\bar{y}$ represents the sample means of the same variables, and n is the number of data points in the dataset. When using this coefficient for feature selection, there are a couple of adjustments that are being considered. For one, the absolute value of the coefficient is considered instead of the exact value. The values of r range from -1 to 1, where -1 indicates a strong negative correlation, 1 indicates a strong positive correlation, and 0 indicates no correlation. When doing feature selection, we want to identify the features with a strong relationship with the class, and it does not matter whether it is positive or negative as long as that feature improves the classification accuracy. Therefore, when selecting the features, if a feature has a high correlation with the class, that means they have a stronger relationship and can be considered as a more important feature. Second, we can only calculate r for numerical values. When we have categorical variables such as the ones we use as labels in our dataset, they have to be first converted into nominal values. Nominal values are discrete values that do not have any numerical relationships. Therefore, when calculating r, each value of the nominal variable is considered a separate category, and a binary indicator variable is created for each value. For example, in our case, we have OS and TOR as classes, which act as indicator variables. Therefore, for every data point, the value of each indicator variable is set to 1 if that variable is present and 0 otherwise. Then the individual correlation is calculated for each indicator variable, and finally, the weighted average is taken. The weight is usually proportional to the frequency of the indicator variable in the dataset [19].
- **Fisher Score** is also a widely used metric in supervised learning which simply tells how much information a feature can reveal about the class (e.g., OS/TOR). The Fisher Score can be defined as below [20] and [21].

$$F_k = \frac{\sum_{i=1}^{c} n_i(\bar{x}_{ki} - \bar{x}_k)^2}{\sum_{i=1}^{c} n_i(\sigma_{ki})^2}, \quad (4)$$

  where F_k is the Fisher Score of the k^{th} feature, c is the number of classes, n_i is the number of instances in the i^{th} class, $\bar{x}_k$ is the mean of the k^{th} feature in all classes, and $\bar{x}_{ki}$ and σ_{ki} are the mean and variance of the k^{th} feature in the i^{th} class, respectively. In the above Equation 4, $\sum_{i=1}^{c} n_i(\bar{x}_{ki} - \bar{x}_k)^2$ refers to the between-class scatter of the k^{th} feature, which is the variation of data points within each class. If this is high, it means the classes can easily be separated. Likewise, $\sum_{i=1}^{c} n_i(\sigma_{ki})^2$ represents the within-class scatter. If this number is low, it means that the data points of the class are situated close to each other and easy to separate. Overall, a high Fisher Score implies that a feature can be used to easily separate classes and hence can be used to identify the most important features.

IV. ANALYSIS

For all experiments, we divide the dataset percentage wise 80:20 into training and testing. Then we do a grid search with 10-fold cross-validation on the training dataset, and evaluate the best model (best hyperparameter combination) on the testing dataset. We used the Scikit Learn (https://scikit-learn.org/stable/) library for machine learning and Pandas (https://pandas.pydata.org/) for data analysis and manipulation.

TABLE 2. Performance of machine learning classifiers to distinguish Onion Service traffic from other Tor traffic.

Experiment	Classifier	Accuracy	Precision	Recall	Training Time (s)	Testing Time (s)
Original (No Defence)	KNN	99.8	**99.9**	99.9	**0.09**	23.03
	RF	99.5	99.7	99.6	1.5	**0.02**
	SVM	**99.9**	**99.9**	**1.0**	32.47	2.11
WTFPAD Defence	KNN	93.5	96	94.6	**0.1**	22.14
	RF	98.9	97.4	98.1	13.82	**0.05**
	SVM	**99.3**	**99.5**	**99.4**	199.57	15.3
TrafficSliver Defence	KNN	82.6	87.2	89.1	**0.16**	175.96
	RF	89.6	91.4	94.4	19.24	**0.2**
	SVM	**91.3**	**93.4**	**94.6**	8493.55	525.12

A. CLASSIFYING ONION SERVICE TRAFFIC FROM OTHER STANDARD TOR TRAFFIC

Here, we answer RQ1: Is it possible to classify Onion Service Traffic from other standard Tor traffic? To identify the classifiability of Onion Service traffic from other Tor traffic, we used the TOR and OS datasets. It gave us a dataset of 136,503 samples, labelled either as TOR or OS, depending on the original dataset. We used three machine learning algorithms on the dataset, namely, K-Nearest Neighbor (KNN), Random Forests (RF), and Support Vector Machine (SVM), and evaluated their performance. We have provided more information on these algorithms in Section II. Table 2 shows the results we obtained when we evaluated the classifiers with the testing dataset.

For evaluating the machine learning classifiers, we used a few metrics, including *accuracy* ($\frac{TP+TN}{TP+TN+FP+FN}$), *precision* ($\frac{TP}{TP+FP}$), *recall* ($\frac{TP}{TP+FN}$), *training time*, and *testing time*. Here, TP, TN, FP, and FN refer to true positives, true negatives, false positives, and false negatives, respectively. Generally, if the dataset is balanced (the same number of samples available for each class), the *accuracy* is enough to provide a good assessment of a classifier's performance. However, as the TOR-OS dataset we use here is imbalanced, we are using *precision* and *recall* as additional metrics to assess the performance. The reason for this is that *precision* and *recall* are not affected by the number of true negatives, which can provide misleading insights in an imbalanced data setting. In Table 2, we show results for classiying Onion Service traffic from other Tor traffic by using all fifty features we have extracted.

From Table 2, we can get the answer to our first research question - RQ1 from the experiments we did with the original (no defence) dataset. We can clearly see that **Onion Service traffic is highly distinguishable from other Tor traffic.** All three classifiers we used performed with 99% *accuracy*. It could be argued that such good accuracy is not surprising, given the clear differences that are present in the datasets for each class of traffic. Non machine learning-based algorithms would likely provide for similar outcomes, with the machine learning classifiers we have listed adding improvement at the 10% level (see later discussion). The importance of our machine learning-based approach is perhaps more forthcoming in the next section, where we discuss the same problem under more difficult circumstances.

B. IDENTIFYING TRAFFIC CLASSES WHEN WEBSITE FINGERPRINTING DEFENCES DEPLOYED

Here, we answer RQ2: How do our results for RQ1 hold when we use modified Tor traffic? In order to obtain answers to our second research question, we carried out the next set of experiments. To recall, in RQ2, we intend to find how certain modifications to Tor traffic, produced by state-of-the-art Website Fingerprinting defences, namely, WTFPAD [10], and TrafficSliver [12], affect Onion Service traffic classification. We used the simulated datasets we created using the WTFPAD and TrafficSliver techniques - our experimental process is as follows.

We first combined the WTFPAD-TOR and WTFPAD-OS datasets to create the WTFPAD dataset. As the WTF-PAD technique uses a padding mechanism, it does not change the number of samples of the original TOR and OS datasets. Next, we combined the TrafficSliver-TOR and TrafficSliver-OS datasets to create the TrafficSliver dataset. As TrafficSliver uses a traffic-splitting strategy, the number of samples increased in this scenario. We configured Traffic-Sliver to split a trace into three sub-traces. We also used the *Batched Weighted Random* splitting strategy, which is the best splitting strategy suggested in [12]. After obtaining the sub-traces, we removed those with a small number of packets (as they could not be used to extract some of our features). As a result, our TrafficSliver dataset consisted of 376,763 traffic traces (271,778 TOR traces and 104,985 OS traces). We ran the same experiments (except for the ones with the least important features) on the WTFPAD and the TrafficSliver datasets. Table 2 shows our results for these experiments. We can obtain several insights by analysing these results in detail.

1) WTFPAD

SVM and RF show a slight reduction ($\sim$0.6%) in *accuracy* on the WTFPAD dataset, while KNN shows a considerable reduction of more than 6%. Both RF and KNN show a larger reduction in *precision* and *recall* compared to SVM. From these results, we can say **WTFPAD does reduce the classifier performance to some extent but still allows them to distinguish Onion Service vs standard Tor traffic successfully**. This behaviour is intuitive given that WTFPAD does padding to both Onion Service and standard Tor traffic. Although it may change the number of packets and some inter-arrival timing statistics, those changes apply to both types of traffic, keeping their differences intact.

2) TRAFFICSLIVER

When we ran our experiments on the TrafficSliver dataset, we observed a significant difference compared to all the previous results we obtained. For KNN, RF, and SVM, the *accuracy* drops are ∼17%, ∼10%, and ∼8%. It implies that the **TrafficSliver modification is actually capable of reducing the identifiability of Onion Service traffic from standard Tor traffic.** However, we can also observe that the *recall* values stay above 90%. It is a result of the models predicting TOR more often than OS. As TrafficSliver splits the traffic into multiple sub-traces, these sub-traces can have contrasting timestamps to the original TOR-OS dataset. Therefore, we can assume that the split traces for OS and TOR have similar characteristics to a certain extent. In addition, we should mention that this performance reduction occurs despite the number of samples increasing (and that the large increase in training and testing times is a consequence of this high number of samples).

C. FEATURE IMPACTS ON TRAFFIC CLASSIFICATION

Here, we answer RQ3: What features impact Onion Service traffic classification most, and what level of performance do they provide? We can answer the first part of RQ3 by looking at Figure 2. From Figure 2a, we can notice that four features, namely, **mean (35) and median (39) outgoing packet concentrations** along with the **2 packet percentage features (49, 50)**, have more information gain with the two classes compared to other features (see Section III for more information about the features and numbers associated with them). Similarly, Figure 2b shows that the above four features, along with the **minimum (36) and SD (38) of packet concentrations**, have a high correlation with the class labels, while Figure 2c confirms those results using Fisher Scores. These are the top six features that seem to have a greater relationship with the traffic type. All three of our feature selection metrics filter out six features out of the fifty. Our next step is to investigate the impact of these features on the performance of the classifiers we used to evaluate RQ1 and RQ2.

1) WHAT LEVEL OF PERFORMANCE DO DIFFERENT FEATURES PROVIDE FOR RQ1?

In Table 3, we show the results for all three datasets we obtained when we only used the top six features we identified (Features **35, 36, 38, 39, 49**, and **50**). In addition, we have used the least important features with the original (no defence) dataset to obtain further insights into their performance impact. These least important features consist of 6 inter-arrival time statistics (Features **2, 3, 4, 10, 11, 12**). The main objective of this experiments is to get an overall sense about the performance of the classifiers, regardless of the input features.

We can see some interesting insights on how the classifier performance change with different (top and bottom) features from Table 3. We can see that the top six features we selected contribute to the majority of the classification ability of the classifiers. Almost all classifiers can classify Onion Service traffic with similar performance with only the top six features (all other features seem redundant). However, we can see that the RF classifier relies more on the feature set compared to the other two classifiers. RF's *accuracy* drops by 9.2% when using the least important features instead of the most important ones. For KNN, this number is only 1.6%, and for SVM, it is 4%. These observations confirm that while the features we have used here play a major role in a classifier's performance, the classifiers themselves have a large impact on the overall result. We can also conclude that using all fifty features is redundant for RQ1.

2) WHAT LEVEL OF PERFORMANCE DO DIFFERENT FEATURES PROVIDE FOR RQ2?

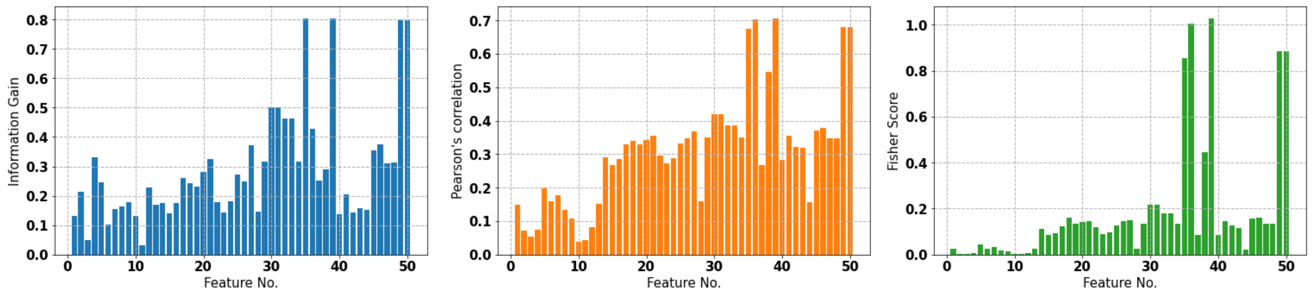
In Table 4, we have presented the performance of each classifier on the modified traffic with only the top six features. For WTFPAD, when we only use the top six features, we can see a more visible reduction in performance than what we saw earlier with all fifty features. Previously, we noticed that the top six features are sufficient to provide an almost ∼99% *accuracy*, *precision*, and *recall* on the original (no defence) dataset. However, when implementing the WTFPAD modifications top six features are not enough to reach that almost perfect classifier performance. Still, all the classifiers show *precision* and *recall* of more than 95%, which is quite good.

When it comes to TrafficSliver, we can observe a drastic change in performance when we use only the top six features. Both *accuracy* and *precision* values for all classifiers drop below 80% in this scenario, while the *recall* values stays more than 90%. This observation is quite similar to what we observed in Table 2. Overall, we can conclude that the top six features are not sufficient to classify Onion Service traffic from other Tor traffic, when the modifications to the traffic are implemented.

D. DETERMINING ONION SERVICE TRAFFIC CHANGES WHEN DEFENCES APPLIED

Next, we combine three datasets, OS, WTFPAD-OS, and TrafficSliver-OS, into a single dataset and label each trace with either ORIG (for original), WTFPAD, and TrafficSliver, respectively. This new dataset consists of 187,991 traces. The main reason for doing this experiment is that we want to determine how similar these modified traces are to the original Onion Service traffic. As this dataset has three different classes, we have to carry out a multi-class classification experiment to evaluate it.

Table 5 shows the *precision* and *recall* for each individual class when we use all fifty features, and when we use the top six important features we mentioned in Section III. As all classes have a different number of samples (traces), *precision* and *recall* are the best metrics for evaluation. From this table, we can get three important insights; (i) Overall, compared to KNN and RF, SVM has a slightly better ability to classify



(a) Information Gain (b) Pearson's Correlation (c) Fisher Score

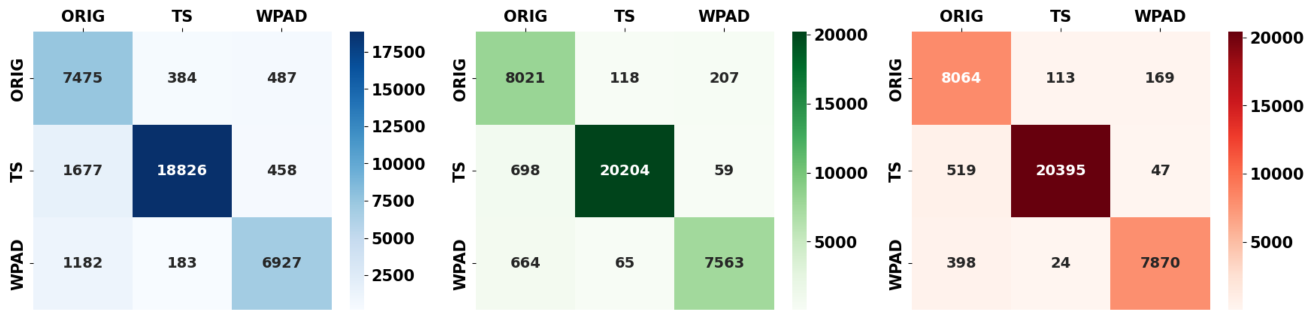
FIGURE 2. Feature Selection Results for TOR-OS combined dataset. Features 1-13 are based on Inter-Arrival Times. Features 14-26 are based on the flow duration. Features 27-34 are based on the number of packets. Features 35-39 are based on packet concentration. Features 40-44 are based on packet frequency. Features 45-48 are based on the packet order. Features 49 and 50 are based on the packet percentage.

TABLE 3. Performance of machine learning classifiers on the original (no defence) dataset with subsets of features.

Dataset	Classifier	Accuracy	Precision	Recall	Training Time (s)	Testing Time (s)
Top 6 Features	KNN	98.9	**99.5**	98.9	**0.12**	1.06
	RF	98.8	99.4	98.8	0.31	**0.009**
	SVM	**99.0**	99.4	**99.1**	23.40	2.23
Bottom 6 Features	KNN	**97.3**	**98.8**	97.3	**0.14**	1.24
	RF	89.6	91.9	93.4	0.46	**0.009**
	SVM	95.0	94.3	**98.8**	165.83	19.09

TABLE 4. Performance of machine learning classifiers with top six features when defences are applied.

Experiment	Classifier	Accuracy	Precision	Recall	Training Time (s)	Testing Time (s)
WTFPAD	KNN	**94.2**	**95.8**	**95.9**	**0.16**	1.15
	RF	95.5	97.5	**95.9**	2.89	**0.03**
	SVM	**96.7**	97.8	**97.4**	107.73	14.9
TrafficSliver	KNN	76.5	**79.8**	90.3	**0.81**	35.26
	RF	77.8	79.9	92.4	4.62	**0.08**
	SVM	**77.9**	79.8	**92.8**	12042.14	673.66



(a) KNN (b) RF (c) SVM

FIGURE 3. Confusion Matrix of different classifiers to distinguish various traffic types.

these traffic types in a multi-class setting. (ii) TrafficSliver traffic seems to have very distinctive features from others, which supports our previous observations in Table 2 and Table 4. (iii) The features play a more important role in this classification task relative to the binary classification experiments we discussed earlier. We can see that the *precision* and *recall* values dropped by more than 20% in some cases when only the top six features were used.

We also provide the confusion matrices for the three classifiers (with all fifty features) in Figure 3, which helps us to get a better understanding of the similarities between the different traffic types. Out of 37,599 testing samples, there are 8,346 Original OS samples, 20,961 TrafficSliver samples, and 8,292 WTFPAD samples. By observing Figure 3, we can see that all traffic types are largely identified. This suggests that **when modifications are used, Onion Service traffic**

TABLE 5. Distinguishability of Onion Service traffic with different modifications.

Experiment	Metric	Classifier	Orig	TS	WPAD
All Features	Prec	KNN	72.3	97.1	88
		RF	85.5	99.1	96.6
		SVM	89.8	99.3	97.3
	Recall	KNN	89.6	89.8	83.5
		RF	96.1	96.4	91.2
		SVM	96.6	97.3	94.9
Top 6 Features	Prec	KNN	68.1	96.1	75.9
		RF	67.4	96.4	73.8
		SVM	71.5	93.8	79.9
	Recall	KNN	75.4	93.9	72.1
		RF	74.2	94	70.9
		SVM	76.2	95.2	71.6

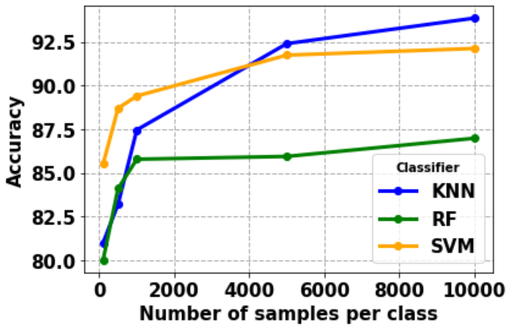


FIGURE 4. TOR-OS experiment with the six least important features and different number of samples per class.

changes significantly from its original version. Another important observation from Figure 3 is that WTFPAD modified traffic are more frequently mis-classified as original Tor traffic compared to TrafficSliver modified traffic, confirming our earlier observations and conclusions.

V. DISCUSSION

A. INSIGHTS

Earlier, we mentioned that traffic flows through six nodes in an Onion Service circuit, while in a standard Tor circuit, this number is three. This can cause a difference in the packet latency, which is captured by the feature set we use. Intuitively, the time-related features that should have a higher classification ability among the features we used. However, the top six features in our experiments are not directly related to time, but the number of packets. This shows that although the outcome of our experiments is as expected, the reasons behind them are not a direct cause of our initial intuition. However, we cannot disregard that intuition completely as the six least-important features, all consisting of IATs, also provide a significant *accuracy*, *precision*, and *recall* when it comes to identifying Onion Service traffic.

In addition, we argue that there are three main reasons for the good results we report for RQ1. These are, the features we have selected, the number of samples in the datasets we have used, and the optimized classifiers we use. In order to support this argument, we conducted a supplementary experiment in which we vary the number of samples we use with the least important features for the TOR-OS dataset. Figure 4 shows the results of this experiment. Here, we can clearly observe that if we did this experiment with subpar features, we would not have obtained 95% *accuracy* even with 10000 samples per class.

B. RUN TIMES

It is perhaps important we discuss the impact of training and testing times (refer to Table 2). When it comes to training time, we have carried out our experiments in an offline setting, i.e., we are not training the classifier in real time. Therefore, if our techniques were to be used in a real-world application, the training would be carried out offline. Therefore, the training time may not have much of an impact in a real-world setting. However, if the dataset is extremely large and the system requires frequent training, it will be useful to employ a technique with a low training overhead. KNN shows the best training time figures for our problem, while SVM shows the worst.

When we consider the testing time, which is representative of the time taken to predict the class of a previously unseen traffic trace, we argue that it is vital to have a low value for that. To further elaborate, let us assume a system designed to alert Onion Service activity inside a company. Such a system has to monitor all the network traffic going out of the company and flag any suspicious traces. If the prediction time of a single traffic capture takes considerable time, that system will either require lots of resources (e.g. computing power, memory) or go into overload. Out of the classifiers we have used, RF shows the best testing times, which in some cases are more than 100 times better than the other classifiers. Therefore, RF would definitely be the ideal candidate to be used in a real-time application. We can also see another interesting observation among the testing times. This is that KNN has higher testing times than training times, which is quite striking in Table 2. The reason for this is that the computational complexity for KNN predictions is a function of the number of samples and the number of features. That is why when we reduce the number of features, the testing time drops drastically. This result shows why it is important to find top-performing features for our classification problem.

C. IMPACT

As we previously mentioned, traffic classification is useful for traffic shaping and traffic monitoring (linking user activity and blocking anonymous traffic). In our case, as we are trying to classify anonymous traffic, it is more relevant for traffic monitoring. As Tor is a censorship circumvention tool, some governments try to restrict access to Tor. If Tor traffic (and Onion Service traffic, for that matter) can be identified easily, then it is easy to implement techniques to restrict that traffic. Also, if there are institutions and businesses that deal with sensitive information and want to restrict access

TABLE 6. Hyperparameter Tuning: Optimum values used for experiments related to RQ1 and RQ2.

Classifier	Hyperparameter	RQ1	RQ2
KNN	n_neighbors metric	4 manhatten	3 euclidean
RF	max_depth n_estimators	5 4	10 10
SVM	C kernel	100 rbf	100 rbf

to the dark web[2] from their computers but do not require to restrict overall Tor access, having techniques to isolate Onion Service traffic can be a great starting place to cater for that requirement. Furthermore, the results from our second research question shed some light on the validity of this work when modifications are introduced to the Tor traffic. Our new results suggest that the results of prior works on Tor traffic classification might be invalid ([3], [6]) in the event these modifications are done to the Tor traffic. Finally, our results highlight that it is important to consider the overall classifiability of Tor traffic when developing (Website Fingerprinting) defences to mask information leakage in Tor traffic.

D. SELECTION OF CLASSIFIERS AND FEATURES

We used insights from existing literature when selecting our classifiers and features. For example, Lashkari et al. [3] used KNN and RF in their work to classify Tor traffic from non-Tor traffic. In addition, most of our features have been previously used by Hayes and Danezis [13] for a Website Fingerprinting attack. By evaluating the related literature, we decided to use these classifiers and features, which ultimately gave good results for our specific problem. We also had to carry out experiments to identify the best hyperparameters for our models. In Table 6, we have mentioned some of the hyperparameters we selected. It should be noted that we tested other additional parameters, but the best values we obtained for them turned out to be the default values used in the Scikit Learn library. Therefore, we have not mentioned them here.

E. SCALABILITY

In this study, we used a relatively large dataset consisting of 136,503 data samples for the first experiment and 513,266 samples for the second. In Figure 4, we showed that our results actually get better with more samples. Therefore, we can argue that our techniques are scalable.

F. FUTURE WORK

Now that we have established Onion Service traffic is distinguishable, and the modifications can reduce this distinguishability, we mention potential future work associated with this line of research. First, the impact of similar modifications on some of the related work could be investigated. For example, although Lashkari et al. [3] show that Tor traffic can be easily differentiated from non-Tor traffic, we argue that the performance of their classifiers may be reduced significantly if modifications in WTFPAD or TrafficSliver are installed in Tor. This argument applies to all other related work mentioned in Table 1. Second, evaluation of how these modifications affect the fingerprintability of individual Onion Services could be attempted. These modifications were originally proposed as Website Fingerprinting defences, and the original papers carried out experiments to show that these changes to Tor traffic do reduce the fingerprintability of websites. However, none of these latter works attempt to investigate the impact those changes have on the fingerprintability of Onion Services. This issue may be worth further investigation.

[2]dark web simply refers to Onion Services.

VI. CONCLUSION

In this work, we answered three research questions focused on Onion Service traffic classification. We evaluated the applicability of supervised machine learning models to classify Onion Service traffic from other Tor traffic. We extracted fifty features from each traffic trace and used that feature set as input to the machine learning classifiers. Our results showed that KNN, RF, and SVM classifiers have the ability to distinguish Onion Service traffic from Tor traffic with a 99% accuracy. Then, we tried to identify whether state-of-the-art Website Fingerprinting defences affect the classifiability of Tor traffic. These defences introduce different modifications to try and obfuscate information leakage from traffic, and we evaluated how those changes affect the Onion Service traffic classification. Our experiments showed that the above classifiers, combined with our feature set, reduce the performance for Onion Service traffic classification. However, we observed that the modified Tor traffic is still distinguishable. Moreover, we used three feature selection metrics, namely, information gain, Pearson's correlation, and Fisher Score, to identify the top features for this task. Those top features were able to provide >98% success for classifying Onion Service traffic from Tor traffic. However, they could not provide such good results when modified Tor traffic traces were used.

REFERENCES

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proc. 13th USENIX Secur. Symp. (SSYM)*, San Diego, CA, USA, Aug. 2004, pp. 303–320.

[2] M. Al Sabah, K. Bauer, and I. Goldberg, "Enhancing Tor's performance using real-time traffic classification," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, Oct. 2012, pp. 73–84.

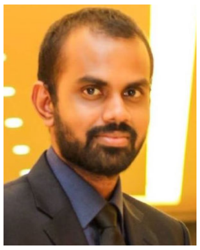
[3] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Tor traffic using time based features," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, Porto, Portugal, Feb. 2017, pp. 253–262.

[4] M. Kim and A. Anpalagan, "Tor traffic classification from raw packet header using convolutional neural network," in *Proc. 1st IEEE Int. Conf. Knowl. Innov. Invention (ICKII)*, Jeju Island, South Korea, Jul. 2018, pp. 187–190.

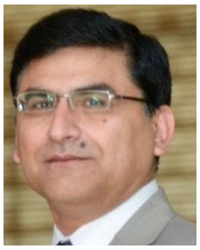
[5] G. He, M. Yang, J. Luo, and X. Gu, "Inferring application type information from Tor encrypted traffic," in *Proc. 2nd Int. Conf. Adv. Cloud Big Data (CBD)*, Washington, DC, USA, Nov. 2014, pp. 220–227.

[6] A. Montieri, D. Ciuonzo, G. Aceto, and A. Pescapé, "Anonymity services tor, I2P, JonDonym: Classifying in the dark (web)," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 3, pp. 662–675, May 2020.

[7] (May 2017). *WCry Ransomware Analysis*. Accessed: Apr. 26, 2023. [Online]. Available: https://www.secureworks.com/research/wcry-ransomware-analysis
[8] (Jul. 2019). *Keeping a Hidden Identity: Mirai C&Cs in Tor Network*. Accessed: Apr. 26, 2023. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/keeping-a-hidden-identity-mirai-ccs-in-tor-network/
[9] (Nov. 2014). *Global Action Against Dark Markets on Tor Network*. Accessed: Aug. 4, 2020. [Online]. Available: https://www.europol.europa.eu/newsroom/news/global-action-against-dark-markets-tor-network
[10] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Proc. 21st Eur. Symp. Res. Comput. Secur. (ESORICS)*, Heraklion, Greece, Sep. 2016, pp. 27–46.
[11] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *Proc. 26th USENIX Secur. Symp. (SEC)*, Vancouver, BC, Canada, Aug. 2017, pp. 1375–1390.
[12] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko, "TrafficSliver: Fighting website fingerprinting attacks with traffic splitting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, Nov. 2020, pp. 1971–1985.
[13] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *Proc. 25th USENIX Conf. Secur. Symp. (SEC)*, Austin, TX, USA, Aug. 2016, pp. 1187–1203.
[14] X. Bai, Y. Zhang, and X. Niu, "Traffic identification of Tor and web-mix," in *Proc. 8th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, Kaohsiung, Taiwan, vol. 1, Nov. 2008, pp. 548–551.
[15] O. Berthold, H. Federrath, and S. Köpsell, "Web MIXes: A system for anonymous and unobservable Internet access," in *Proc. Int. Workshop Design Issues Anonymity Unobservability*, in Lecture Notes in Computer Science, vol. 2009, H. Federrath, Ed., Berkeley, CA, USA, Jul. 2000, pp. 115–129.
[16] B. Zantout and R. Haraty, "I2P data communication system," in *Proc. 10th Int. Conf. Netw. (ICN)*, Sint Maarten, The Netherlands, Jan. 2011, pp. 401–409.
[17] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, Oct. 2018, pp. 1928–1943.
[18] R. Overdorf, M. Juárez, G. Acar, R. Greenstadt, and C. Díaz, "How unique is your.onion?: An analysis of the fingerprintability of Tor onion services," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Dallas, TX, USA, Oct. 2017, pp. 2021–2036.
[19] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2011.
[20] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Vancouver, BC, Canada, Dec. 2005, pp. 507–514.
[21] M. Gan and L. Zhang, "Iteratively local Fisher score for feature selection," *Appl. Intell.*, vol. 51, pp. 6167–6181, Aug. 2021.



ISHAN KARUNANAYAKE (Graduate Student Member, IEEE) received the B.Sc. degree in electronic and telecommunication engineering from the University of Moratuwa, Sri Lanka, in 2018. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, University of New South Wales (UNSW), Australia. He is affiliated with the Cyber Security Cooperative Research Centre (CSCRC), Australia. His research interests include network security, malware analysis, anonymity and privacy, and machine learning.



NADEEM AHMED received the M.S. and Ph.D. degrees in computer science and engineering from the University of New South Wales, Sydney, NSW, Australia, in 2000 and 2007, respectively. He was the Head of the Computing Department, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Pakistan. He is currently a Senior Research Fellow with the Cyber Security Cooperative Research Centre (CSCRC), Australia. His research interests include cyber security, the IoT, wireless sensor networks, and software-defined networking.

ROBERT MALANEY (Senior Member, IEEE) received the B.Sc. degree in physics from the University of Glasgow, and the Ph.D. degree in physics from the University of St Andrews, Scotland. He has previously held research positions with Caltech, University of California, Berkeley, Berkeley, CA, USA, and with the University of Toronto. He was a farmer Principal Research Scientist with CSIRO. He is currently a Professor with the School of Electrical Engineering and Telecommunications, University of New South Wales, Australia. He has more than 200 publications.

RAFIQUL ISLAM is currently an Associate Professor with the School of Computing and Mathematics, Charles Sturt University, Australia. He was leading the Cybersecurity Research Team and has developed a strong background in leadership, sustainability, and collaborative research in the area. He has a strong publication record and has published more than 170 peer-reviewed research papers, book chapters, and books. He has a strong research background in cybersecurity with a specific focus on malware analysis and classification, authentication, security in the cloud, privacy in social media, the IoT, and dark web. He is an Associate Editor of the *International Journal of Computers and Applications* and a Guest Editor of various reputed journals.

SANJAY K. JHA (Senior Member, IEEE) is currently a Full Professor with the School of Computer Science and Engineering, University of New South Wales (UNSW), Australia, where he leads the Cybersecurity Cooperative Research Centre. He is a Chief Scientist and the Research Director of the UNSW Institute for Cybersecurity (IFCYBER). He is the principal author of the book *Engineering Internet QoS*. His research interests include network security, wireless mesh and sensor networks, and the Internet of Things. His editorial affiliations include the IEEE TRANSACTIONS ON MOBILE COMPUTING and the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING. He is a Co-Editor of the *book Wireless Sensor Networks: A Systems Perspective*.