Fingerprinting encrypted network traffic types using machine learning

Nathan Meheus

Supervisors: Prof. dr. ir. Bart Dhoedt, Prof. dr. ir. Pieter Simoens
Counsellors: ing. Sam Leroux, Ir. Steven Bohez

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Bart Dhoedt
Faculty of Engineering and Architecture
Academic year 2016-2017
Fingerprinting encrypted network traffic types using machine learning

Nathan Meheus

Supervisors: Prof. dr. ir. Bart Dhoedt, Prof. dr. ir. Pieter Simoens
Counsellors: ing. Sam Leroux, Ir. Steven Bohez

Master’s dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Bart Dhoedt
Faculty of Engineering and Architecture
Academic year 2016-2017
Preface

During this academic year, I received help from a lot of people, which made me able to successfully finish this thesis. This way, I would like to thank them for their support.

First and foremost, I would like to thank my supervisors, Sam Leroux and Ir. Steven Bohez, for their continuous helpfulness, for taking the time to assist me on a weekly basis, for their constructive feedback and for sharing their expertise on the topic with me.

My promotors, prof. Bart Dhoedt and prof. Pieter Simoens, I would like to thank for the opportunity to perform research on this topic. As well as for their feedback during the mid-term evaluations.

Furthermore I would like to thank Ir. Andy Van Maele for his assistance with setting up a network experiment, and Dr. Brecht Vermeulen for his willingness to help when I had trouble working on the iMinds Virtual Wall.

And last but not least, I would like to thank my family and my girlfriend for their immense support and endless understanding.

Nathan Meheus, June 1, 2017
Permission for use of content

“The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In the case of any other use, the copyright terms have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.”

Nathan Meheus, June 1, 2017
Abstract

Confidentiality, privacy and anonymity are highly valued by Internet users. While protocols providing these services are capable of obfuscating the content and/or headers of IP packets, they are often neglecting other packet characteristics. This work will explore some of these characteristics, and show the remaining vulnerabilities by extracting the application type from an encrypted connection. We will focus on two such protocols, being IPsec and Tor. Our experiments show that classifying encrypted network traffic can be done with high accuracy in a limited setting. But we will extend these experiments to observe the effect of more realistic network traffic on the achieved accuracies.

Keywords

Network traffic analysis, IPsec, Tor, machine learning
Fingerprinting encrypted network traffic types using machine learning

Nathan Meheus

Supervisor(s): Sam Leroux, Ir. Steven Bohez

Promotor(s): prof. Bart Dhoedt, prof. Pieter Simoens

Abstract—Confidentiality, privacy and anonymity are highly valued by Internet users. While protocols providing these services are capable of obfuscating the content and/or headers of IP packets, they are often neglecting other packet characteristics. In this work, we explore some of these characteristics, and show the remaining vulnerabilities by extracting the application type from an encrypted network connection. We focus on two such protocols, being IPsec and Tor. Our experiments show that classifying encrypted network traffic can be done with high accuracy in a limited setting.

Keywords—Network traffic analysis, IPsec, Tor, machine learning

I. INTRODUCTION

SECURITY and privacy of network users are important topics in computer networking. Various methods to ensure these aspects of network traffic have been developed, and their usage has been increasing rather rapidly [1]. We distinguish two different approaches when evaluating these methods. The encrypting approach, by which the transmitted data packets are transformed/encrypted into hard-to-reveal ones. And the anonymizing approach, which has as main goal to keep the identity of the user hidden. However, sometimes it is necessary to extract some information from traffic streams. For example, the network administrator of a school or an organization might want to block certain applications for safety purposes. In those cases, traffic analysis can come in handy. Traffic analysis can be defined as the process of monitoring the behavior of communication traffic for the sake of discovering useful patterns inside the transmitted packets [2]. Even with the currently used methods for protecting network traffic, there is still some useful information that can expose the traffic behavior and user identities. This kind of information is not concealed by the protecting protocol, which means it can be utilized to analyze the traffic. It should be clear that traffic analysis can be used for malicious purposes as well, such as recognizing user identities to violate their private communications [2].

This paper starts with providing an overview of fingerprinting methods used in literature, followed by the approach taken for this work. In sections 3, 4 and 5, we discuss the experiments for three different scenarios. In the first scenario, no encryption is in place, except for the application specific encryption measures. The second scenario deals with IPsec encryption. And finally, a more advanced encryption and anonymizing protocol called Tor is investigated. The last section of this paper summarizes the obtained results.

II. RELATED WORK

Network traffic analysis has gathered a lot of interest, which resulted in quite some available literature. We omit all research dealing with active fingerprinting of network traffic, and focus only on the passive approaches which do not interfere with the packet streams. Since encryption protocols obfuscate the content (and in some cases also the header) of packets, deep packet analysis is not an option for fingerprinting packet streams. This is why most research on passive fingerprinting techniques goes out to finding stream characteristics that can be used to identify distinct applications, webpages,... Those characteristics are all related to two aspects of packet streams, being the size and the timing. When dealing with a machine learning problem, these characteristics are referred to as features. From now on, we will adopt this terminology. The packet size and packet inter-arrival time (IAT) features can be used as they are, or in a aggregated fashion. A commonly used variant of the packet size is the packet size combined with the direction (upstream or downstream), resulting in a negative packet size for the downstream packets and a positive packet size for the upstream packets. This feature has been proven useful by Gang Lu et al. [3]. Features can also be observed at an aggregated level of multiple packets. A burst is such a possible aggregation, and is defined as a sequence of packets sent in one direction that lie between two packets sent in the opposite direction [4]. Both the timing and the sizes of bursts can be useful. Another interesting aggregation that is proposed in literature is the “Surge Period”. A Surge Period marks the parts of a traffic trace where the channel is continuously busy transmitting packets upstream or downstream [4]. Other more complex techniques exist as well, such as protocol fingerprinting, which is statistically formed with the packet size, inter-arrival time and packet order [3].

Not only the used features contribute to the performance of network traffic fingerprinting, also the machine learning techniques that are used matter. Liberatore and Levine used a naive Bayes classifier to predict the visited web page, out of a set of 2000 different pages, based on the packet size combined with the direction [5]. Another popular classifier that was used is the decision tree. Using the packet size and IAT, Gang Lu et al. obtained an accuracy of 86.636% for classifying 5 different applications with a C4.5 decision tree [3]. The support vector machine (SVM) classifier was used by A. Panchenko et al. for website fingerprinting of 775 different websites, visited using a Tor browser. The used features were packet size, direction and the total number of transmitted bytes. They were able to achieve...
a recognition rate of 55% [6]. Also J. Luo et al. applied the SVM to a Tor website fingerprinting problem with 100 different websites. Using both size and timing features at burst level they obtained 65% accuracy. This overview is by no means exhaustive, many more methodologies and results can be found in the article by Petr Velan et al. [1].

III. PROBLEM DEFINITION AND APPROACH

The problem at hand is identifying the application(s) that are using an encrypted link. The possible applications that we consider are VoIP, BitTorrent, HTTP browsing and YouTube video streaming. The goal of this work is twofold. We first examined two different feature spaces, packet size in combination with packet inter-arrival time being one, and burst size and timing being the other. And Secondly, we compared different encryption protocols. Since no network traces of desired quality are publicly available, we generated our own network captures. For every traffic type, we chose a fixed procedure to produce the network traffic. A VoIP trace starts just before starting or receiving a voice call, en goes on for a brief moment of the actual call. The BitTorrent traces start when we added a torrent file to the torrent client, and some time of the download. The HTTP browsing covers a Google search and following one of the suggested links. And the YouTube traffic consists of surfing to the YouTube homepage, clicking a video link and watching the video play for some time. Depending on the investigated protocol, a different environment was used for the gathering of the data. These different environments will be discussed in the corresponding sections.

The general approach and the used feature spaces do remain the same for all scenarios. The approach is schematized in figure 1.

The raw data consists of the complete network traces, 10 for every application under examination. Every trace was split into trace windows, containing 1024 packets. This was done for two reasons. First of all it increases the amount of data that is available for our machine learning problem. And secondly, a windowed approach mimics a real time approach, meaning that we could use the classifier after every 1024 packets that are captured. The feature extraction includes transforming the per packet (or per burst) extracted characteristics to a uniform format to be used as input for the machine learning models. The way this was done is by constructing a 2D histogram from the extracted values. We used logarithmic scales (natural logarithm) to focus more on the order of magnitude of the features than on their actual values. Every dimension was divided into 32 bins, resulting in a total of 1024 features. Normalization takes place per sample, by converting the histogram to a 2D probability density function. The packet size and IAT feature space of a VoIP (Skype) and a BitTorrent window are shown in figure 2 and figure 3 respectively.

These features were extracted from an unencrypted network.
link. Similar figures can be produced for the other applications, the other feature space and the other encryption scenarios. The machine learning models that we used for the experiments are a naïve Bayes classifier, a logistic regression classifier and a random forest. The hyperparameters of all these models were tuned using 4 fold cross validation. For every experiment, a new model was trained. This means that a model that is trained on one type of encrypted data is not used for another type.

IV. UNENCRYPTED NETWORK TRAFFIC

The device that was used for capturing unencrypted network traffic, is an Asus R500VD-SX198V, with Ubuntu 16.04 LTS as operating system, connected to a wireless 802.11n network. The tool that was used for capturing is Wireshark 2.0.5, VoIP traces were captured while running Skype version 4.3.0.37, the torrent client that was used is Deluge 1.3.12 and Mozilla Firefox 50.0 served as browser for obtaining HTTP and YouTube traces. Caching is disabled, and all applications are forced to use IPv4 only. All applications were ran in an isolated network namespace, such that the captured traces only contain packets from the targeted application. After windowing, 468 samples were obtained, from which 20% was kept aside as the test set. After tuning the three different classifiers, accuracies ranging from 94.55% up to 100% were obtained. All of the obtained metrics are summarized in table I. The table also contains the results for when both feature spaces are used as input for the classifiers. This results in a small improvement over the two single feature spaces separately.

![Normalized confusion matrix](image)

Fig. 4. Confusion matrix for the random forest classifier with the packet size & IAT feature space.

A closer look at the results reveals that the classifier confuses some HTTP samples with YouTube samples and vice versa. This is shown using a confusion matrix for the random forest classifier with the packet size and IAT feature space in figure 4. This behavior can be explained by the fact that YouTube traffic is just a specific case of HTTP traffic.

Not only the hyperparameters of the models have an effect on the performance, also the window size and the number of bins used for constructing the histograms can have an impact. Within the tested range, varying the number of bins had little effect on the achieved accuracies, however, the probabilities for the predictions are lower—thus less certain—for a smaller amount of bins. A similar effect is observed when varying the window size. The uncertainty of the predictions rises as the window size goes down, up to a certain point, when the windows get to small to extract useful features.

### TABLE I

<table>
<thead>
<tr>
<th></th>
<th>NB</th>
<th>LR</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>94.55</td>
<td>99.09</td>
<td>97.27</td>
</tr>
<tr>
<td>Burst features</td>
<td>96.36</td>
<td>99.09</td>
<td>97.27</td>
</tr>
<tr>
<td>Both</td>
<td>97.27</td>
<td>100.00</td>
<td>99.09</td>
</tr>
</tbody>
</table>

Starting with unencrypted network provides us with a baseline which we can validate for both IPsec and Tor in the next sections.

V. IPSEC-ENCRYPTED NETWORK TRAFFIC

A. Experimental setup

The IPsec experiment environment is the iMinds Virtual Wall [7]. In this environment, we set up an experiment with two nodes, an IPsec server and an IPsec client. The IPsec client is again ran from an isolated network namespace. An IPsec tunnel is established between the network namespace and the server, and the server acts as default router for the client. This results in all traffic going to or coming from the public Internet to be routed through the IPsec tunnel. IPsec supports a wide variety of configurations [8]. In our setup, we opted for tunnel mode with Encapsulated Security Payload (ESP) to provide both encryption and authentication. The same applications as in the unencrypted scenario were used, but the Firefox version was 52.0 this time.

B. Results

The feature spaces for both unencrypted network traffic and IPsec-encrypted network traffic show very similar visualizations. Therefore, results comparable to those of the previous section were expected. The results for all three classifiers and for the three different inputs are summarized in table II, and show indeed close resemblance to the results that were obtained for unencrypted traffic.

### TABLE II

<table>
<thead>
<tr>
<th></th>
<th>NB</th>
<th>LR</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>93.55</td>
<td>95.70</td>
<td>96.77</td>
</tr>
<tr>
<td>Burst features</td>
<td>87.10</td>
<td>94.62</td>
<td>93.55</td>
</tr>
<tr>
<td>Both</td>
<td>94.62</td>
<td>95.70</td>
<td>96.77</td>
</tr>
</tbody>
</table>

1NB: Naive Bayes
LR: Logistic regression
RF: Random forest
The achieved results can be explained by the operation of IPsec. Since both packet size and timing determine the feature spaces, it is useful to understand the effect of IPsec on these characteristics.

The timing is only influenced by the time needed for the encryption/decryption and authentication check of the packets. In our configuration, IPsec uses the Advanced Encryption Standard (AES), with a 128-bit key for encryption. This encryption is combined with a SHA-256 as the hash function to provide data integrity and authentication. The delays caused by AES-128 and SHA-256 are neglectable, since both have very efficient implementations.

The size of the packets is altered by the protocol specifications for ESP packets and the use of tunnel mode. Tunnel mode introduces and additional IP header (typically 20 bytes) because of the encapsulation. The format of an ESP packet adds another 8 bytes of protocol parameters and 32 bytes for the integrity check value, which is a SHA-256 hash. ESP also supports padding, but this is disabled in all default configuration, and therefore also in our experiment.

The reasons that our classifiers can cope with this are both the fact that all packets are affected by approximately the same addition of information, and the use of a logarithmically scaled packet size axis.

C. Background traffic

So far we have only considered the ideal situation in which all captured packets originate from a single application, by running that application in an isolated namespace. In a more realistic scenario however, several background processes may be running and sending/receiving packets as well. Because the iMinds Virtual Wall is a complex research environment, the background packet rate is high, approximately 50,000 packets/second (this includes both upstream and downstream packets). The number of background packets, observed in a realistic home environment (in which the unencrypted network traffic traces were captured) is approximately 20 packets/second. Another observation about the characteristics of the background traffic is the rather small packet sizes that occur, typically smaller than 150 bytes. A script was used to mimic this background traffic characteristics. Using this approach, we examined the effect of different packet rates. Since the generated packets contain random bytes, the receiving end does not reply on them. In order to fully resemble a realistic scenario, the packet generation script had to be ran on both the server and the client node. The results for the packet size and IAT feature space as input for the logistic regression classifier are shown in figure 5.

Although the global trend is obvious, some irregularities need some explaining. As expected, the classification accuracy drops as the packet rate increases. This is easily explained by the fact that for a higher packet rate, a window (which represents a single sample) contains less application specific packets and more background traffic packets, thus complicating the classification process. The irregularities are due to the random nature of the traces, and are only a couple of percents. The human interaction involved in for example the generation of HTTP traces can have a big impact on the content of the windows, since this depends heavily on the speed with which the user clicks links. A smoother graph might also be obtained if the experiment was done multiple times and the average accuracy over these experiment was plotted instead of the accuracy of a single experiment.

D. Multi-label classification

Up to this point we have always assumed that only one of the four applications was running at each time instant. We have however considered the possibility that we capture some background traffic packets. A next step to making the experiment more comparable to a realistic scenario is the use of multi-label classification. This means that every sample can have one up to four labels, which is something that might occur in a realistic environment. A person can watch YouTube videos while downloading content via a Torrent client for example.

Due to the nature of the chosen feature spaces, being packet size & IAT or burst characteristics, the histograms for multi-labeled samples do not resemble the addition of their single-labeled components. Although packet sizes might show this behavior, inter-arrival times do not. They tend to shorten as packets from different applications interleave each other. Also burst from one application may be interrupted by packets from the other application.

The data in this experiment reuses the single-labeled data, but additional data with multiple labels was gathered. This additional data consists of ten traces for every possible combination of two, three or four applications. Splitting the data for the train set and the test set is done such that both sets have approximately the same ratio of single-labeled/multi-labeled samples. A problem that occurs in this experiment is the uncertainty on the true label of a sample. It is possible that both the BitTorrent application and the HTTP browsing are ongoing at the same time, but due to the higher packet rate of the BitTorrent traffic, a window might only contain BitTorrent packets while still being labeled with two labels. We hoped to reduce this issue by starting the capturing process after the applications have started, for example, after a VoIP call is initiated, and no longer during the actual calling phase.

The results for the random forest classifier are summarized in figure 5. The effect of background traffic on the classification accuracy.

![Fig. 5. The effect of background traffic on the classification accuracy.](image-url)
in table III. Since the accuracy in multi-label classification is a harsh metric (it does not take into account partially correct predicted samples), we included the Hamming loss, which does exactly that.

**TABLE III**

**COMPARISON OF CLASSIFIER PERFORMANCE FOR THE DIFFERENT FEATURE SPACES**

<table>
<thead>
<tr>
<th>Feature Space</th>
<th>Accuracy [%]</th>
<th>Hamming loss [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>83.87</td>
<td>4.62</td>
</tr>
<tr>
<td>Burst features</td>
<td>76.83</td>
<td>6.89</td>
</tr>
<tr>
<td>Both</td>
<td>83.58</td>
<td>4.62</td>
</tr>
</tbody>
</table>

Based on the Hamming loss metric and the fact that some sample might actually have an incorrect true label, we can see that the random forest classifier is able to cope with multiple applications running simultaneously. Once again, the effect of background traffic was examined, and in the case of multi-label classification, a more severe drop in performance was noticed. Intuitively this is explained by the limited information left within a single window, since with high background traffic rates and multiple applications, every window only contains very few packets for each of the running applications.

**VI. TOR-ENCRYPTED NETWORK TRAFFIC**

A. Tor methodology

The second encryption and anonymizing protocol that we have experimented with is The Onion Router (Tor). Tor is more complex than IPsec since it uses a network of Tor relay nodes to provide anonymity to its users. It does so by routing all traffic over three relay nodes, before it is routed to its actual destination. Each relay node only knows the previous and the next node, resulting in anonymity. To provide data confidentiality, Tor establishes a TLS connection between the client and every relay node. This produces several layers of encryption, comparable with the layers of an onion, hence the name. A schematic overview of a Tor circuit is shown in figure 6.

B. Experimental setup

This experiment was also performed on the iMinds Virtual Wall. The Tor experiment consisted of three nodes. A client, to run the applications, which we call the workstation in order to be consistent with the terminology of the Tor topology that we set up, a gateway, which acts as a proxy for the workstation, and a router, which is the default router for the gateway. The Tor topology that we have used is called an isolated proxy. All applications on the workstation node are forced to use the gateway as SOCKS-proxy, applications that do not provide this possibility can not be used with this concept. The gateway node acts as the Tor client, and sends all packets coming from the workstation over the Tor circuit. The router node is added to provide a link on which can be captured without obtaining packets from the environment’s control interface. The applications for this experiment are the same as for the IPsec experiment.

C. Results

The results for all three classifiers and for the three different inputs are summarized in table IV. The classification is less accurate in this experiment, compared to the previous ones. The accuracy drops 10 to 15%.

These results are again explained by looking at the impact of Tor on the packet sizes and timing characteristics. The effect on the timing is twofold. First of all, there is the longer path that is taken by Tor because of the three relay nodes through which all packets are routed. This circuit changes about every 10 minutes [10], thus resulting in different delay in different network traces. The second possible effect on the timing is again due to the encryption/decryption of the packets. Tor uses TLSv1.2, which itself uses primarily AES as its encryption protocol and SHA-1 as its message authentication code (MAC). This is very similar to IPsec, and is neglectable as we stated in section V-B. The effect of Tor on the packet sizes is caused by the TLS encryption by each of the relay nodes which make up the circuit. Every node adds an additional layer of TLS security. The structure of a TLS record results in an additional 21 bytes (containing protocol parameters and the SHA-1 MAC) per relay node to each packet. Since Tor uses three relay nodes to construct a circuit, a packet—at the client—is 63 bytes longer than the unencrypted packet. This is comparable to the increase in packet size caused by IPsec. It is thus the additional routing delay, introduced by the anonymizing aspect of Tor, which is the most important cause of the reduced accuracy.

**TABLE IV**

**COMPARISON OF DIFFERENT CLASSIFIERS FOR TOR-ENCRYPTED TRAFFIC**

<table>
<thead>
<tr>
<th>Feature Space</th>
<th>NB</th>
<th>LR</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>80.56</td>
<td>77.78</td>
<td>84.72</td>
</tr>
<tr>
<td>Burst features</td>
<td>86.11</td>
<td>80.56</td>
<td>86.11</td>
</tr>
<tr>
<td>Both</td>
<td>83.33</td>
<td>80.56</td>
<td>80.56</td>
</tr>
</tbody>
</table>

2 An average Hamming loss of 4.62% means that, on average, 4.62% of the predicted labels are incorrect (both false negatives and false positives).
VII. CONCLUSIONS

We have shown that size and timing features, at both individual packet level and at aggregated burst level, can be successfully used to fingerprint encrypted network traffic types. A first baseline on unencrypted network traffic traces provided some understanding on potential difficulties regarding the confusion of HTTP browsing and YouTube video streaming traffic. These difficulties reoccurred when dealing with IPsec and Tor encryption. Other than that, IPsec did not alter the features significantly and thus similar accuracy measures were obtained. Tor on the other hand does have an effect on the timing characteristics of a network stream, because of the circuit of relay nodes that it uses in order to provide anonymity. This resulted in a noticeable accuracy drop. We do however believe that more data, captured from different Tor circuits can make our models more robust against the delay introduced by Tor. Most of the experiments made use of an isolated network in order to separate the targeted application from all other traffic. We also introduced artificial background traffic, which showed that an increased packet rate caused a drop in accuracy. This effect is however very moderate if we consider that a background packet rate of approximately 20 packets/second was observed in a realistic scenario. A last scenario that we considered was the possible simultaneous presence of multiple applications. The achieved accuracy metrics were lower than when dealing with single-labeled data, but these should be discussed with caution since the true labels might not be fully correct.

ACKNOWLEDGMENTS

The author would like to sincerely thank his supervisors Sam Leroux and Ir. Steven Bohez for their support and expertise given during the performed experiments and their feedback on previous versions of this work.

REFERENCES

# Contents

1 Introduction

1.1 General machine learning approach ........................................... 2
  1.1.1 Training, validation and testing ........................................ 2
  1.1.2 Preprocessing, feature extraction and normalization ................. 4
  1.1.3 Performance metrics ...................................................... 4
  1.1.4 Complete flow ......................................................... 6

2 Literature review

2.1 Introduction ................................................................. 7

2.2 Encrypting and anonymizing protocols ................................. 8
  2.2.1 Encrypting protocols ................................................... 8
  2.2.2 Anonymizing protocols ............................................... 9

2.3 Different trace characteristics ....................................... 10
  2.3.1 Packet size ........................................................... 10
  2.3.2 Packet inter-arrival time .......................................... 11
  2.3.3 Packet count and packet ordering .................................. 11
  2.3.4 Burst size ............................................................. 11
  2.3.5 Surge period .......................................................... 11
  2.3.6 Consumed bandwidth ............................................... 12
  2.3.7 Others ................................................................. 12
  2.3.8 From packet feature to flow feature ................................ 12

2.4 Machine learning techniques and results ......................... 13
  2.4.1 Bayesian approaches ............................................... 13
  2.4.2 Decision tree ....................................................... 13
  2.4.3 Support vector machines .......................................... 14
3 Unencrypted network traffic analysis

3.1 Data gathering ................................. 18
   3.1.1 Publicly available traces ................. 18
   3.1.2 Self-generated traces .................... 19
       3.1.2.1 Wired or wireless captures .......... 21
3.2 Feature spaces ............................... 22
   3.2.1 Feature extraction ....................... 22
   3.2.2 Packet size and inter-arrival time ........ 23
   3.2.3 Burst time and burst size ............... 24
3.3 The classification problem ..................... 25
   3.3.1 Preparing the input ...................... 25
   3.3.2 Classifiers ............................... 26
       3.3.2.1 Naive Bayes ......................... 26
       3.3.2.2 Random forest ....................... 27
       3.3.2.3 Logistic regression ................. 28
       3.3.2.4 Multi-layer perceptron ............. 30
   3.3.3 Other parameters .......................... 31
3.4 Conclusion .................................... 31

4 IPsec-encrypted network traffic analysis

4.1 Experimental setup .......................... 32
   4.1.1 iMinds Virtual Wall ................. 33
   4.1.2 IPsec: strongSwan ...................... 34
4.2 Classification of IPsec traffic ................. 35
   4.2.1 The data .................................. 35
4.2.2 Feature spaces .............................................. 35
4.2.3 Classifiers .................................................... 37
4.2.4 Analysis of the results .................................. 38
4.3 A non-ideal environment .................................. 39
4.3.1 Characteristics of background traffic .................. 39
4.3.2 Effect of background traffic ......................... 40
4.4 Conclusion ....................................................... 42

5 Multi-label classification of IPsec-encrypted network traffic 43
5.1 The data ......................................................... 43
5.2 Feature spaces ................................................ 44
5.3 Classifiers ...................................................... 47
5.4 A non-ideal environment .................................. 48
5.5 Conclusion ....................................................... 49

6 Tor-encrypted network traffic analysis ............. 50
6.1 The Onion Router ............................................. 50
6.2 Experimental setup .......................................... 51
6.3 Classification of Tor traffic ............................... 53
6.3.1 The data ...................................................... 53
6.3.2 Feature spaces ............................................ 54
6.3.3 Classifiers .................................................. 56
6.3.4 Analysis of the results ................................ 57
6.4 Conclusion ....................................................... 58

A The trained logistic regression classifier ............ 59
A.1 Unencrypted network traffic ......................... 60
A.1.1 Packet size and inter-arrival time .................... 60
A.1.2 Burst features ............................................ 62
A.2 IPsec network traffic ...................................... 64
A.2.1 Packet size and inter-arrival time .................... 64
A.2.2 Burst features ............................................ 66
A.3 Tor network traffic ........................................ 68
A.3.1 Packet size and inter-arrival time .................... 68
Chapter 1

Introduction

Based on the title, this thesis can be divided into two parts. The first aspect is “fingerprinting encrypted network traffic types”. Intuitively this means extracting the traffic type, or the running application(s), from a network stream that has been encrypted. A more complete explanation of this problem is given in chapter 2. The second aspect contained in the title of this thesis is “using machine learning”. Machine learning is defined as “programming computers to optimize a performance criterion using example data or past experience” [1]. In this case, our network traces are the examples, and we would like the computer to learn how to correctly predict the type of such a network trace. The correctness of its predictions is the performance criterion to optimize. Since this is a very vague definition, we will go over some of the fundamentals in the remainder of this introductory chapter. These best practices will be used throughout the rest of this work. The next chapter will not only discuss the fingerprinting problem, it will also give an overview of the available literature on this topic. Afterwards, the chapters bring both the theoretical aspects of the literature review as well as the discussed machine learning approach into practice by elaborating on the performed experiments. The chapters tackle the problem in order of increasing complexity. In chapter 3, we start with establishing a baseline on unencrypted network traffic, which we validate for IPsec encrypted network traffic in chapter 4. We expand the IPsec experiment to a multi-label classification problem in chapter 5. And finally, we study the effects of a more complex encryption scheme called “Tor” in chapter 6.
1.1 General machine learning approach

Machine learning covers a wide variety of solutions, so it would be very impractical to try to cover them all in this work. We will only discuss the machine learning branch called classification, in which the goal is to predict a label (or multiple labels), coming from a finite, discrete set, for a given sample, based on the labeled samples which the program learned from. The knowledge that the program extracts from the data is stored in a model, by means of the model parameters. At this time, it is not important to define the wide variety of models and their (mathematical) foundations that exist. It is however important that the parameters of these models are learned from the data. There are two types of parameters, the internal parameters, which are directly learned from the data, based on the mathematical foundation of the model, and the hyperparameters, which can for example determine how well the model can cope with noisy data. This second set of parameters is not mathematically extracted from the data, but they are tuned according to the performance metric by the implementer of the machine learning solution. An example is a regularization parameter that penalizes more complex models because simpler models are more likely to be correct than more complex models. Another example is the tolerance on the stopping criteria, training stops once the next iteration of training does not provide an increase of the performance bigger than the stopping criteria. In what follows, we will discuss how the data is prepared for the process, how these hyperparameters are determined, and what metrics are used for evaluating the solution.

1.1.1 Training, validation and testing

A machine learning problem requires data in order to “solve” it. In the preceding section, we made no distinctions between the different purposes of the data. However, in a machine learning problem, there is the need for data to solve the problem, being the training data, data to tune the hyperparameters, the validation data, and finally, data that is only used for evaluating performance, the test data.

In its simplest form, a machine learning problem is tackled by training a model on the training data with a specific set of hyperparameter values, and validating its performance on the validation set. This is done as many times as there are combinations of hyperparameter values that we wish to test. After this tuning phase, we know the optimal set of hyperparameters, their values are the ones that resulted in the best performing model on the validation set. Then, the
1.1 General machine learning approach

The model is retrained with its best hyperparameters, using both the training set and the validation set as input. The final performance of the model is then obtained by applying the model to the test set.

The reason that this approach is taken, and that the hyperparameters are not simply determined based on the performance on the test set is the problem of data leakage. When we would not use a validation set, and test every combination of hyperparameters on the test set, we would be using the test set for “training” as well. This is not desirable, as the test set represents unseen data, which has as only purpose validating the performance of the final model.

Splitting all available data in three sets means reducing the actual amount of data to train the model on. To overcome this, a different hyperparameter tuning approach is available, called k-fold cross validation. The data is in this case only split into two parts initially, the training set and the test set. Determining the hyperparameters goes as follows. The number of folds (k) is chosen, typically between 4 and 10, and the training set gets split in k parts. The same approach as with a standard validation set is used, but this time each of the k parts act as validation set once. The obtained performance measures get averaged over the folds, and again the optimal hyperparameters are obtained. A schematic overview of this process is shown in figure 1.1.

Throughout our experiments, we will be using a k-value of 4.

Figure 1.1: K-fold cross validation. The number of folds is 5 in this example [2].
1.1.2 Preprocessing, feature extraction and normalization

In many cases, the raw data cannot be used as input for the classification models. Some preprocessing is required. This preprocessing step depends heavily on the data that is available. It often involves some kind of data cleaning, for example, removing Address Resolution Protocol (ARP) packets from the network traces.

When the data is preprocessed, it is still not fully ready to be used as input. The useful characteristics, called features have to be extracted from the data. For each data point (a sample), these features have to be obtained. Doing so, one converts the raw data to an $m \times n$ matrix, with a row for each of the $m$ samples, and a column for each of the $n$ features. What those features are, depends on the problem at hand. The feature used in this thesis will be discussed in chapter 3.

Finally, we discuss the normalization procedure. The performance of some machine learning algorithms is affected by the (difference in) scale of the features. That is why normalization is a necessity. The parameters for normalization are always determined based on the training data, and not on all data. If one would determine these parameters based on all data, data leakage would occur. The way that normalization is applied in this case will also become clear in chapter 3.

1.1.3 Performance metrics

In order to evaluate a model, a certain performance criterion is needed. When dealing with single-label classification (each sample belongs only to one class), accuracy is the most often used metric. The average accuracy is defined as the number of correctly classified samples divided by the total number of classified samples.

\[
\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}
\]

In multi-label classification, a sample can belong to more than one class. Using the accuracy in this case would not take into account partially correct predicted labels. An alternative metric doing exactly this is the Hamming loss. The Hamming loss is the fraction of labels that are incorrectly predicted. So instead of striving for a model with maximum accuracy, we strive for one with minimal Hamming loss. The formula is as follows:
1.1 General machine learning approach

\[ L_{Hamming} = \frac{1}{N} \sum_{n=1}^{N} \left( \frac{1}{n_{classes}} \sum_{i=1}^{n_{classes}} 1(y_n \neq \hat{y}_n) \right) \]

The function \( 1(y_{n,i} \neq \hat{y}_{n,i}) \) evaluates to 1 if the i’th label of the prediction for sample n differs from the actual i’th label of sample n, and to 0 otherwise.

Apart from these two commonly used metrics, there are some visualizations that are often used for evaluating the model performance. The first one is a confusion matrix, that can only be used when dealing with single label classification.

By definition, entry i, j in a confusion matrix is the number of observations actually with label i, but predicted to have label j. Sometimes, these confusion matrices are normalized. This is done by dividing each entry of a row by the sum of that row. Doing so, the actual sample count in the matrix entries is converted into fractional counts, which tend to give a better insight.

A second alternative is more like a summarization of a number of different performance metrics, and is called a classification report. The structure is shown in table 1.1.

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>label 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>label 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>label 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg / total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The precision for a certain label is the percentage of correctly predicted samples compared to the total said label predictions. Intuitively, this is the ability of the classifier not to label a sample with a certain label while this label does not hold for this sample. The recall is the percentage of samples for which the label is correctly predicted compared to the total amount of samples that actually should have had this label predicted. Intuitively, this is the ability of the classifier to find all the samples with a certain label. That f1-score is the harmonic mean of the recall and the precision. And the support is the number of occurrences of a certain label in the test set.
1.1.4 Complete flow

A schematic representation of the complete flow is shown in figure 1.2.

Figure 1.2: The complete machine learning approach.
Chapter 2

Literature review

2.1 Introduction

Security and privacy of network users have been, and still are, hot topics. Various methods to ensure these aspects of network traffic have been developed, and their usage has been increasing rather rapidly [3]. Two approaches can be considered when evaluating these methods. The encrypting approach, by which the transmitted data packets are transformed/encrypted into hard-to-reveal ones. And the anonymizing approach, which has as main goal to keep the identity of the user hidden. However, sometimes it is necessary to extract some information from traffic streams. For example, the network administrator of a school or an organization might want to block certain applications for safety purposes. In those cases, traffic analysis can come in handy.

Traffic analysis can be defined as the process of monitoring the behavior of communication traffic for the sake of discovering useful patterns inside the transmitted packets [4]. Even with the currently used methods for protecting one’s network traffic, there is still some useful information that can expose the traffic behavior and user identities. This kind of information is not concealed by the protecting protocol, which means it can be utilized to analyze the traffic. Examples of information of this nature are packet lengths, timing information, packet directions, etc. It should be clear that traffic analysis can be used for malicious purposes as well, such as recognizing user identities to violate their private communications [4].

In literature, traffic analysis has been discussed in various ways. A common distinction that is made, is application fingerprinting vs website fingerprinting. As the terms state, the first option
is used to distinguish between different applications that are being employed within a network, while the alternative identifies individual websites that are being visited. This is sometimes called Precise Page Identification (PPI) [5]. Another possible separation between analysis techniques is the passive approaches vs the active ones. When applying an active analysis technique, the analyzing entity (such as an ISP) cannot only observe packets passively but also manipulate them actively, by delaying certain packets for example [6]. These techniques are not used very often, and won’t be further discussed.

The remainder of this chapter will first go more into depth on the best-known encryption and anonymizing protocols, because their characteristics determine which features can still be exploited for traffic analysis. Afterwards, these features are discussed. And once those are covered, the actual fingerprinting techniques will be discussed, as well as the experiments in which these results were obtained. This chapter will be wrapped up with the major conclusions that can be drawn from literature.

2.2 Encrypting and anonymizing protocols

As mentioned in the introduction, we distinguish two options to protect data packets. Encrypting the payload, or anonymizing the packet stream. Both options are covered in the next subsections by exploring the general method and providing some examples.

2.2.1 Encrypting protocols

Encryption protocols, such as SSL/TLS, IPsec, SSH,... generally operate in two phases, the initialization of the connection and the transport of encrypted data. During the first phase, an initial handshake takes place, which determines the versioning and other algorithm specific parameters. After that, authentication takes place and secret keys for encryption are established. These keys are then used for encrypting data in the second phase.

Some methods that are available in literature require packets from the first phase to be captured. The goal of capturing these, not yet encrypted, packets is to extract information about the encryption parameters that will be used. This can provide insight in for example the effects on the packet sizes after encryption [5].

Within the group of encryption protocols, another distinction can be made, based on which
2.2 Encrypting and anonymizing protocols

The OSI reference model is a framework that divides communication into layers to facilitate the exchange of information. Layer 1 is the physical layer, which deals with the physical transmission of data. Layer 2 is the data link layer, responsible for connecting two devices and ensuring they can communicate. Layer 3 is the network layer, which is responsible for routing data from one network to another. Layer 4 is the transport layer, which ensures reliable delivery of data. Layer 5 is the session layer, which manages the connection between users. Layer 6 is the presentation layer, which deals with the representation of data. Layer 7 is the application layer, which allows users to interact with the network.

IPsec on the one hand, operates on the network layer, and is able to not only protect the data within an IP-packet, but also the header itself. This makes traffic analysis based on port numbers or IP-addresses inapplicable. SSL/TLS and SSH on the other hand are only responsible for protection at the transport layer. Together with application specific protection protocols, as used by Skype, BitTorrent, . . . , they only protect the packets at higher level and thus create opportunities for traffic analysis based on header information. When there has not been a single form of encryption applied, deep packet analysis can be used to obtain information from the actual data payload.

![Figure 2.1: The OSI model.](image)

### 2.2.2 Anonymizing protocols

Anonymizing protocols are used to hide the identities of their users to prevent them from being traced when using the Internet. This service is often required to obtain content which would otherwise not be accessible (blocked by the ISP, illegal, . . . ). Another scenario in which online anonymity is desirable is sharing data in regions in which Internet access is restricted by the government. The term anonymizing networks is often used instead of protocols, because network traffic is routed through a network of routers that would not be visited when using standard routing protocols. The best-known options for anonymizing network traffic are virtual private networks (VPNs), web proxies and The Onion Router (Tor) [7].
VPNs allow users to spoof their physical location by replacing their own IP address by an IP address given by the VPN provider. The user is disguised as being physically at the location of the VPN provider, thus its real IP address cannot be tracked. All traffic from and to the user, is first routed through the VPN.

Web proxies work in a very similar way, all web requests en responses go via a proxy server that hides the true identity of the client.

The last, and most powerful option is Tor. Tor utilizes a worldwide network of volunteer routers (relay agents) to hide the true identity of its users. As is the case for most VPNs, Tor also uses encryption to offer its users maximum Internet privacy. In Tor, this is done at every relay agent, hence the name The Onion Router, because of the analogy with the multiple layers of an onion.

2.3 Different trace characteristics

Literature describes a lot of different characteristics (features) that can be used to classify network traffic. These range from simple ones, such as packet size, to more complex options, like protocol fingerprinting, which are often just complex derivations of the basic characteristics, aggregated over (a part) of the flow. The next sections will briefly discuss the majority of features that are used for traffic analysis.

First, it should be noted that an encrypted network trace can easily be represented by a sequence of pairs \((t_k, s_k)\), \(k = 1, 2, \ldots\) where \(t_k\) is the time at which the k’th packet is observed and \(s_k\) is the size of the observed packet. The sign of \(s_k\) indicates whether the packet is traveling in the uplink or downlink direction. This representation provides the same information as the encrypted traces itself, because the payload of the packets is not accessible anyway. This notation will be used throughout the rest of this thesis.

2.3.1 Packet size

The size of a packet is directly linked to the contained payload, which follows an application specific profile, and can thus be used for classifying said application. The direction of the packet is in most cases considered together with the size, which results in the sign of \(s_k\).
2.3.2 Packet inter-arrival time

Another feature that is often used is timing information. The main drawback of using packet sizes as feature is the ease by which privacy providing protocols can influence this feature. By padding the encrypted payload with a pseudorandom number of bytes, no more useful information can be extracted from the packet size. The timing of packets is less adaptable, since it would have a direct impact on the performance of the web service [8]. In contrast to packet size, timing information is a continuous variable, so it is often discretized in literature [9].

2.3.3 Packet count and packet ordering

Packet count and ordering can be useful features, when the boundaries of the traffic captures are very well know. The ordering of the packets in combination with the amount of packets within a web page fetch can be very specific for a certain website, for example. But the features are no longer useful if the trace boundaries do not match the web page fetch boundaries. In practice this is often the case, so this combination of features is only useful in a theoretical ideal setting.

2.3.4 Burst size

In literature, a burst is defined as a sequence of (non-acknowledgement) packets sent in one direction that lie between two packets sent in the opposite direction. The bandwidth of a burst is the total size of all packets contained in the burst, in bytes, and the burst count is the number of packets within the burst [10, 11]. This two-folded, aggregated feature describes the properties of the network traffic at a higher level, because it does not only take packet features into account, but also the correlation between packets. A burst can typically be used to identify a web page fetch.

2.3.5 Surge period

Another aggregated feature is surge period. It was first defined by Yan Shi and Subir Biswas as follows [10]. A Surge Period marks the parts of a traffic trace where the channel is continuously busy transmitting packets upstream or downstream. Any packet except for the first one and the last one within the surge period should be separated from its predecessor and subsequent packets by a time period no larger than a predefined time window size [10]. The window size is a
parameter that depends on the network conditions. This feature is based on the same intuitions as the burst size feature, but it is slightly more advanced, since it has some extra parameters. Typically, a fixed percentage of the total number of packets in a trace should be covered by surge periods, and only the N (a predefined number, that indicates how many features are needed) most important surge periods are used.

### 2.3.6 Consumed bandwidth

The bandwidth feature is rather self-explanatory, and it is strongly related with the previously discussed features. In practice, the bandwidth is always observed together with the direction, this is called the per-direction bandwidth [11].

### 2.3.7 Others

A lot of other features are described in literature. Most of them are combinations of the above, such as protocol fingerprinting, which is statistically formed with three properties of IP packets: packet size, inter-arrival time and packet order [9]. Others involve complex transforms, for example the use of the first n components of the Haar Wavelet Transformation. This feature is used to represent coarse-scale variation information of a trace. Each trace is first converted to a bit-rate series using a 100ms time window, then interpolated to 4096 data points before transforming. Only the first 15 components are kept as features [10].

### 2.3.8 From packet feature to flow feature

The number of packets contained in a trace is typically very high, so the number of features that can be extracted from them as well. To reduce this feature space, a couple of techniques are used. A first technique has already been discussed, namely the aggregation of feature in an intuitive fashion, such as bursts or surge periods. An alternative method is more general and uses binning to restrict the number of features. And lastly, using only the first N packets from a trace is also done in literature. The problem with using only certain packets, is that the trace boundaries must be known. If this is not the case, or if windowing is applied, N random packets are used, and it has been shown that this does not yield good results.
2.4 Machine learning techniques and results

Not only the used features contribute to the performance of network traffic fingerprinting, also the machine learning techniques that are used matter. In this section, a short overview of the techniques that are described in literature will be given, accompanied by the obtained result. It is important to state the context in which the results were obtained, because almost every study uses different network content sources, encrypting and anonymizing protocols, ... One should take this into account when comparing the results.

2.4.1 Bayesian approaches

Naive Bayes is an easy and intuitive technique, that often produces good results. The technique is based on Bayes rule, and makes the assumption that input features are conditionally independent given the output class. Because of this assumption, computation of the class probabilities is straight forward. The classification task boils down to assigning the class label of the highest class probability to the feature vector [1].

\[ c = \arg \max_c [p_{c_1}, p_{c_2}, \ldots, p_{c_n}] \]

It is used by Liberatore and Levine to classify traces containing web page fetches, from about 2000 different websites. The data is gathered by forwarding the fetches through a SSH-tunnel to a proxy. The feature that is used is packet size (with the direction embedded) [12]. In the paper, the effects of different parameters are compared, e.g. time between training and testing, number of websites used for training,

2.4.2 Decision tree

Another machine learning method is using a decision tree. The biggest advantage of this approach is the ability to represent it visually. A decision tree is a tree structure in which every node represents a condition (e.g. packet size > 512 bytes). The branches leaving from this node each correspond to a different value that can be assigned to the node (in this example there are two options, since it’s a boolean condition, however, the condition could have been packet size, and the outgoing arrows could corresponds to different packet size ranges). A lot of different algorithms to construct decision trees exist, each with their own characteristics, such as the
2.4 Machine learning techniques and results

criteria that is used to determine the best splitting conditions at the nodes. The decision tree that is used in literature is generated by the C4.5 algorithm [9]. The C4.5 algorithm constructs the tree based on the difference in entropy between the multiple classes, when only taking into account a single feature at each time. The feature that maximizes this difference in entropy (the normalized information gain) is selected as the decision making feature. If no feature happens to provide any information gain, the algorithm moves up in the hierarchy of the tree and creates a node based on the statistical properties of the subset of the training data at that point in the tree. Traffic data for five applications — http, ssl, qq, eDonkey and Xunlei — was collected by the authors. The encryption methods are however not mentioned, so we assume that none were taken. The obtained accuracy on unseen data was maximal when using a combination of packetsize and inter-arrival time for the first two packets of a trace. An accuracy of 86.636% was obtained.

2.4.3 Support vector machines

The basic idea behind support vector machines is finding a hyperplane in the featurespace, which separates datapoints from different classes, and has the largest distance possible to the closest instances of each of the classes. This is called the maximum margin.

Support vector machines are used by A. Panchenko et al. for website fingerprinting. The dataset covers 775 different websites, which are visited using a Tor browser. The used features are packetsize, direction and the total number of transmitted bytes. They were able to achieve a recognition rate of 55% [13].

Also J. Luo et al. have used SVMs for website fingerprinting in a Tor browsing environment. The features that were observed in this paper are both burst information such as timing and size, as well as the total per-direction bandwidth and the total per-direction number of packets. The achieved classification accuracy is 65% for an experiment with 100 websites. It is
important to note that an active fingerprinting technique was used, to stimulate certain packet retransmissions.

2.4.4 Artificial neural networks

The last machine learning technique that will be discussed is the artificial neural network. This network consists of multiple layers of perceptrons, and is thus sometimes called a multi-layer perceptron. Each perceptron has a number of inputs, \( x_j, j = 1, \ldots, d \) with corresponding weights \( w_j \). Every perceptron also has a bias input \( x_0 = +1 \), and a corresponding weight \( w_0 \). If we represent all the inputs as a vector \( x = [1, x_1, \ldots, x_d] \), and all weights as \( w = [w_0, \ldots, w_d] \), we can write the output of the perceptron as \( y = A(w^T x) \). With the \( A \) being a certain activation function. The sigmoid function is an often used example.

![Visual representation of a perceptron.](image)

To create the neural network, the outputs from one layer of perceptrons are used as the inputs for the next. Training the weights is done by using the backpropagation algorithm, which will not be further discussed [1].

P. G. Bridges et al. used this machine learning technique to classify network traffic belonging to one of the following applications: HTTP browsing, chatting, online gaming, video streaming and BitTorrent. They used the packet size, the packet size distribution and the inter arrival time as input features. The obtained classification accuracy is 71.394% [14].
2.4.5 Others

Machine learning approaches are commonly used for traffic analysis, but other options exist. Similarity-based fingerprinting is one of those. In this case, traces are classified based on a certain distance metric, for example the cosine similarity or the cross-correlation metric, instead of being classified by a pre-built model [8, 15].

2.5 Defense techniques

A lot of research has been done to find methods to obscure the network traffic even more, and thus to provide defense against fingerprinting. Although these methods might result in better privacy conditions, they can also decrease the users’ experience since they may directly affect timing, or doing so by affecting the number of transmitted bytes.

2.5.1 Payload padding

The most straightforward approach, as already mentioned briefly in the section about features, is padding of the packet payload. Padding increases the bandwidth requirements, since additional, useless information is appended to the packets. Different flavors of padding exist.

- Random padding: appending a random amount of bytes to the packets (the amount can be fixed for a session, or calculated for each packet individually), while not exceeding the MTU [11].
- MTU padding: padding every packet to the MTU size. In practice, this method makes all features that depend on packet size useless, however, this method dramatically increases overall data transmitted, by up to 150% according to Liberatore and Levine [12].
- Mice-elephants padding: pad each packet to either 100 or MTU. All small packets (mostly ACKs) are padded to one size, and all other packets to another. This is a better performing method compared to MTU padding, since a lot of small packets don’t have to be padded up to the MTU. The overhead is reduced to 50% according to Liberatore and Levine [12].
2.5.2 Camouflaging

Camouflage is a method to intentionally change the patterns of the data traffic. In general, it uses simultaneous network traffic to conceal the application being used. An example that is used for website fingerprinting is loading a randomly chosen web page (background page) simultaneously with the actually requested one. By doing so, the traffic is obfuscated by loading several pages simultaneously [13].

2.5.3 Buflo

Buffered Fixed-Length Obfuscator is a countermeasure which hides the total time, bandwidth use, and burst patterns. It does this by sending fixed-length \((d)\) packets at a fixed interval \((\rho)\) for at least a fixed amount of time \((\tau)\). If the traffic flow needs to continue after this fixed time, it keeps sending fixed-length packets at a fixed interval. If there is no data to send before the fixed time runs out, dummy data is sent instead. Using Buflo results in a high bandwidth overhead. The parameters \(d\), \(\rho\) and \(\tau\) determine the actual impact on the network performance [11].

2.6 Conclusion

Within this chapter, we have formulated the use case at hand, being fingerprinting encrypted network traffic, and sketched some possible scenarios in which this technique could be useful. We started by giving a concise overview of encryption and anonymizing protocols, which is far from exhaustive. Later on, we continued with an enumeration of different trace characteristics that have been used in literature to perform network fingerprinting. All the discussed characteristics are related to the timing and packet size information contained in a network trace, since these aspects of network traffic are rarely properly obfuscated. Also, a brief overview of some of the machine learning algorithms, which have been applied to the fingerprinting problem, is given. For a more in depth understanding of the discussed algorithms, the reader is referred to [1]. We concluded this chapter with some defense techniques that can be used in order to prevent fingerprinting. However, none of these techniques come truly for free, since they manipulate timing and/or size to hide these characteristics.
Chapter 3

Unencrypted network traffic analysis

Introduction

This work starts with the classification of unencrypted network traffic. This is mainly done for two reasons. For comparison purposes in a later phase, and as an introduction to the frameworks for myself. This chapter will cover the experimental setup that was used for gathering data, it will discuss the investigated feature spaces, and finally it will cover the performance of some tested classifiers.

3.1 Data gathering

For the remainder of this chapter, the analyzed traffic types are the following ones: YouTube video traffic, Google search traffic (which we will call HTTP-traffic), Skype voice calls and BitTorrent traffic.

3.1.1 Publicly available traces

Although literature provides a decent amount of information about network traffic classification, there are not a lot of available repositories that can be used for the purposes of this work. The majority of available datasets contain traces for malware analysis, which is not of our interest. However, several community-based data dump websites exist. These websites contain network traces of different kinds, and are usually tagged with some information about the nature of the trace. Since these websites are often not moderated, the quality of the traces is questionable for
our goal. Some issues are uncertainty about caching, capturing environment, device,... Visual analysis of these traces were a motivation to generate our own traces.

Figure 3.1: Scatterplot of two traces from the HTTP class. Every point represents a single packet, based on its packet size (x-axis) and its inter-arrival time (y-axis). Upstream packets have a positive size, while downstream packets have a negative size.

Figure 3.2: Scatterplot of a trace from the HTTP class. The amount of packets that were captured is low. (Same conventions as in fig 3.1).

It can be derived from figures 3.1 - 3.2 that traffic from the same nature is not easily grouped together based on the observed features. Also, the amount of captured packets is highly variable.

3.1.2 Self-generated traces

In order to overcome the shortcomings of publicly available data, the traces that are used for the actual traffic analysis were generated in a controlled environment.

All applications were monitored in an isolated network namespace, such that the captured traces only contain packets from the targeted application.
3.1 Data gathering

Figure 3.3: The usage of a network namespace to isolate applications [16].

The tool that was used for capturing is Wireshark 2.0.5, Skype traces were captured while running version 4.3.0.37, the torrent client that was used is Deluge 1.3.12 and Mozilla Firefox 50.0 served as browser for obtaining HTTP and YouTube traces. All traces were captured with caching disabled. The device that is used is an Asus R500VD-SX198V, with Ubuntu 16.04 LTS as operating system, connected to a wireless 802.11n network.

An overview of the traces used further in this chapter can be found in table 3.1. The numbers are calculated from 10 captured traces of each type.

A Skype trace consists of either calling a person, or being called, some time for the actual conversation, and ending the call. The YouTube traces contain packets from surfing to the YouTube homepage, and opening a random video that is watched until the end. The HTTP traces cover the Google search and opening one of the suggested webpages. And the BitTorrent traces start at the point of opening a torrent file in the torrent client, and goes on for a brief moment of downloading the actual content.
3.1 Data gathering

Figure 3.4: Scatterplot of four traces from the HTTP class. The four traces look very similar. (Same conventions as in fig 3.1).

Table 3.1: Overview of the captured traces.

<table>
<thead>
<tr>
<th></th>
<th>Mean #packets</th>
<th>Max #packets</th>
<th>Min #packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skype</td>
<td>7015</td>
<td>7633</td>
<td>6066</td>
</tr>
<tr>
<td>HTTP</td>
<td>14148.9</td>
<td>36241</td>
<td>3393</td>
</tr>
<tr>
<td>YouTube</td>
<td>13864.3</td>
<td>22040</td>
<td>5918</td>
</tr>
<tr>
<td>BitTorrent¹</td>
<td>14981.4</td>
<td>14984</td>
<td>14980</td>
</tr>
</tbody>
</table>

3.1.2.1 Wired or wireless captures

As stated above, the traces that were used were gathered over a wireless link, however, for completeness, some of the captures were reproduced over a wired link. Visual analysis of both showed no significant differences, so for convenience reasons, only the wireless captured ones were used.

¹The number of packets for BitTorrent traffic does not represent a complete transaction, because the number of packets is extremely large in this case.
3.2 Feature spaces

From the gathered data, two different feature spaces were extracted. Since the details about these features have been discussed in chapter 2, this section will provide visual proof of their usability as input for a machine learning classifier.

3.2.1 Feature extraction

As stated in chapter 2, a certain strategy is needed to convert the packet features to flow features. We choose to use the histogram approach, but after conversion to a logarithmic scale (natural logarithm). This was done to focus more on the order of magnitude of the features than their actual values. Another step that was taken before the actual extraction is windowing. Every capture file was split into traces containing 1024 packets. The goal of this approach is twofold. First of all, it increases the amount of data that is available for our machine learning problem. And secondly, a windowed approach mimics a real time approach, meaning that we could use the classifier after every 1024 packets that are captured. Figure 3.6 shows that a random window from a trace has the same characteristics as the full trace.
3.2 Feature spaces

Figure 3.6: Two 2D histograms representing the packet size inter-arrival time feature spaces of a random window from a Skype trace (right) and the full trace (left).

3.2.2 Packet size and inter-arrival time

From figure 3.7, some observations about the network traffic types can be made. A first observation is the very clear distinction of Skype traffic from the other ones. The Skype protocol results in small packets, approximately equal size in downstream and upstream, and at small inter-arrival times. Another important observation is the similarity between HTTP traffic and YouTube traffic. This can be explained by noting that YouTube traffic makes use of the HTTP-protocol for its video streaming services.
3.2 Feature spaces

Figure 3.7: The 2D-histograms (packet size and IAT) of all four traffic types.

3.2.3 Burst time and burst size

 Visualization of the burst time and burst size feature space can also give some insight in the characteristics of the different traffic types. The first thing noticed is again the difference of the Skype traffic that has been explained before. Also, the similarities between HTTP and YouTube traffic can again be observed.
3.3 The classification problem

In the remainder of this chapter, the details of the actual classification will be discussed. Both
the data and the different classification techniques are covered in the next sections.

3.3.1 Preparing the input

The complete set of data that is available for the problem at hand contains the 40 captured
traces, which result in a total of 468 samples after windowing (1024 packets per window). 20%
of these samples are set aside for performance evaluation. This is the test set. The remaining
80% of the samples are used for training and validation.

Determining the histogram edges is part of the normalization procedure, since these edges need
to be determined based on only the training data, in order to avoid data leakage. The strategy
that was used is finding the maximum and minimum values along all dimensions of the feature
space of the training data, and dividing this range in a fixed number of equal-sized bins. The
bin edges are exported, and used for generating the histograms for the validation and test
data. Normalization takes place per sample, by converting the histogram to a two-dimensional
probability density function.
3.3 The classification problem

3.3.2 Classifiers

3.3.2.1 Naive Bayes

The naive Bayes classifier that was used is the multinomial naive Bayes classifier available in the scikit-learn python library. The documentation states the following:

The distribution is parametrized by vectors \( \theta_y = (\theta_{y1}, \ldots, \theta_{yn}) \) for each class \( y \), where \( n \) is the number of features and \( \theta_{yi} \) is the probability \( P(x_i \mid y) \) of feature \( i \) appearing in a sample belonging to class \( y \).

The parameters \( \theta_y \) is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

\[
\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}
\]

where \( N_{yi} = \sum_{x \in T} x_i \) is the number of times feature \( i \) appears in a sample of class \( y \) in the training set \( T \), and \( N_y = \sum_{i=1}^{\lvert T \rvert} N_{yi} \) is the total count of all features for class \( y \).

The smoothing priors \( \alpha \geq 0 \) accounts for features not present in the learning samples and prevents zero probabilities in further computations [17].

The only hyperparameter to tune in this case is the smoothing parameter \( \alpha \). Using the optimal \( \alpha \)-value—which we determined using 4-fold cross validation—we obtain the classification accuracies that can be found in table 3.2.

The accompanying confusion matrices are shown in figure 3.9.

(a) Size & IAT feature space  
(b) Burst feature space
3.3 The classification problem

3.3.2.2 Random forest

For the random forest classifier, the sklearn implementation was used as well. A random forest classifier is an ensemble of decision trees, in which each tree is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. And only a random subset of the feature space is used for splitting in the tree. The total number of trees in the ensemble is a hyperparameter for this model. The final prediction of the random forest classifier is obtained by averaging the probabilistic predictions of the single decision trees. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model [17].

The only hyperparameter that was tuned, is the number of decision trees that are contained within the random forest classifier. Other hyperparameters that might be considered are the number of features to use for a single tree, the splitting criterion, . . . Using the optimal number of trees—again determined using 4-fold cross validation—, we obtain the classification accuracies that can be found in table 3.2.

The accompanying confusion matrices are shown in figure 3.10.
3.3 The classification problem

(a) Size & IAT feature space

(b) Burst feature space

(c) Combined feature space

Figure 3.10: The confusion matrices for the random forest classifier.

3.3.2.3 Logistic regression

Despite the name, logistic regression is a classification technique. It bases its classification on the ratio of class-conditional densities, \( p(x|C_i) \), where \( i \) is 0 or 1, since logistic regression uses a one-vs-rest scheme. We start by defining:

\[
a = \log \frac{p(x|C_0)}{1 - p(x|C_0)}
\]

If we then approximate \( a \) by a linear model, we get \( a = g(x) = \tilde{w}^T \tilde{x} \) in which \( x \) represents a matrix of containing the input features for all sample points, and \( w \) is the weight vector which
needs to be learned. The learning is done by initializing the weights randomly and adjusting them using gradient descent to minimize cross-entropy error.

The process of optimizing the weights depends on some hyperparameters, of which C and the tolerance are the most important. The C parameter determines the regularization strength, and is used to penalize complexer models, thus avoiding overfitting. And the tolerance parameter is used as the stopping criteria for the gradient descent algorithm. The obtained weights are discussed in more detail in appendix A.

Using the optimal values for these two parameters, we obtain the classification accuracies that can be found in table 3.2.

The accompanying confusion matrices are shown in figure 3.11.

![Confusion matrices](image)

(a) Size & IAT feature space  
(b) Burst feature space  
(c) Combined feature space

Figure 3.11: The confusion matrices for the logistic regression classifier.
3.3.2.4 Multi-layer perceptron

The multi-layer perceptron classifier is an artificial neural network, as described in section 2.4.4. The parameters that determine the strength of an MLP classifier are mainly the number of nodes and the number of layers. For this work, only a very limited amount of options was tested, ranging from three to five layers, with each hidden layer containing 100 nodes. The lack of varying these parameters results in worse outcomes, compared to the other classifiers. This behavior can clearly be derived from both the confusion matrices in figure 3.12, and the summarization in table 3.2.

We do however believe that with a more complete search for optimal parameters, the accuracy of the MLP classifier could be as high as those of the other classifiers.

![Normalized confusion matrix](image)

(a) Size & IAT feature space  
(b) Burst feature space  
(c) Combined feature space

Figure 3.12: The confusion matrices for the multi-layer perceptron classifier.
Table 3.2: Comparison of different classifiers.

<table>
<thead>
<tr>
<th>Accuracy [%]</th>
<th>Naive Bayes</th>
<th>Logistic regression</th>
<th>Random forest</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>94.55</td>
<td>99.09</td>
<td>97.27</td>
<td>91.82</td>
</tr>
<tr>
<td>Burst features</td>
<td>96.36</td>
<td>99.09</td>
<td>97.27</td>
<td>80.91</td>
</tr>
<tr>
<td>Both</td>
<td>97.27</td>
<td>100.00</td>
<td>99.09</td>
<td>94.55</td>
</tr>
</tbody>
</table>

3.3.3 Other parameters

Apart from the hyperparameters, there are some other parameters that can have an effect on the performance of the classifier. During the feature extraction, a choice has to be made for the number of bins (in each dimension) to use when constructing the histograms. All of the above results have been obtained with 32 bins in each dimension. This results in 1024 input features when using packet size and inter-arrival time or burst time and burst size. When the two feature spaces are combined, we obtain a total of 2048 input features. For other values, up to a certain minimum, of the amount of bins, a similar accuracy was obtained. However, the probabilities for the predictions are lower—thus less certain—for a smaller amount of bins. Another parameter that influences the performance of the classifiers is the window size. Again, there is a range of values for which the classification accuracy does not differ significantly, but in which the uncertainty of the predictions rises as the window size goes down.

3.4 Conclusion

This chapter started with discussing the poor quality of publicly available network traces for our purposes, followed by the methodology that was used for generating our own traces. This was done by isolating the applications under investigation from all other processes on the computer. Two feature spaces, firstly burst characteristics and secondly packet size combined with packet inter-arrival time, have been studied and validated using four different classifiers. With very limited hyperparameter tuning, we have obtained accuracy measures ranging from 94.55% up to 100% for three of the four classifiers. The MLP classifier performs a little worse, but this is due to limited hyperparameter testing. The confusion matrices learned us that distinguishing between HTTP and YouTube traffic can be a challenge for our classifiers. These misclassifications are justified by noting that YouTube traffic is in fact a specific case of HTTP traffic.
Chapter 4

IPsec-encrypted network traffic analysis

Introduction

In the previous chapter we have shown that we can obtain a set of features from a packet stream which enables us to categorize this stream into one of the chosen applications, being HTTP, Skype, YouTube and Torrent. Another observation from the previous chapter is the limited amount of data, in combination with the small set of tested hyperparameters, that was used to obtain the achieved accuracies. From this chapter on, we will shift our focus towards encrypted network traffic analysis, starting with IPsec encryption. We will mainly discuss the observed similarities and differences with the scenario from the previous chapter.

4.1 Experimental setup

A first difference that occurs is the change of environment. While all unencrypted traffic was captured on a PC in a typical home setting, the IPsec-encrypted traffic is generated and captured using an iMinds Virtual Wall experiment. This environmental change is due to the fact that the IPsec setup requires two endpoints (a client and a server) between which the encryption is established. We opted for control over both endpoints, thus needing two physical machines. The alternative is to connect to a public IPsec service, in which case a single PC is sufficient. The setup on the iMinds Virtual Wall can be split up in two parts. The first part consists of enabling
IPv4 traffic from the client to reach the public Internet via the server. And in the second part, all traffic between the client and the server gets encrypted.

### 4.1.1 iMinds Virtual Wall

Without additional configuration, the iMinds Virtual Wall is an IPv6 environment. However, so far we have only looked into IPv4 traffic, since Skype does not yet support IPv6. This implies that some configuration is needed to enable NAT’ed IPv4 traffic on the iMinds Virtual Wall [18]. We configured this on the server only, and enabled IPv4-forwarding. An additional configuration that we added to the server is the ability to make use of NAT for the traffic coming from the client. On the client, we added an isolated network namespace to separate the traffic coming from the applications from that originating from the background processes. The default route on the client sends all packets to the server via a direct link.

![Diagram](image.png)

**Figure 4.1:** The setup on the iMinds Virtual Wall, with an isolated network namespace on the client node.

In the final scenario, packets originate from the namespace, travel through the client’s default namespace, to the server. Where they get NAT’ed and sent to the iMinds Virtual Wall router.
(over which we have no control). This setup is visualized in figure 4.1. The shaded blue tunnel between the network namespace and the server represents the IPsec tunnel and is discussed next.

### 4.1.2 IPsec: strongSwan

To enable an IPsec-encrypted tunnel between two endpoints, both endpoints need to run an IPsec service. For this experiment, we used the well-known strongSwan daemon, which relies on a couple of configuration files for setting the IPsec parameters [19].

IPsec can operate in two modes, transport and tunnel mode. The difference is an additional IP header that hides the actual IP address of the sender, if the sender would not be the endpoint itself. This is shown in figure 4.2.

![Figure 4.2: The different IPsec modes.](image)

For this experiment, tunnel mode was used. In order to establish the encrypted connection, an authentication mechanism is needed. The available options are public key signatures, such as RSA-signatures or Elliptic Curve DSA-signatures, or pre-shared keys, which is the option that was adopted for this experiment, since it requires minimal effort to set up the connection.

A last parameter that is worth mentioning is the subset option. This option specifies to the other endpoint what subnet it can reach via itself. At the client, this is its own IP address, while the server advertises the entire Internet, being 0/0 in subnet notation.
4.2 Classification of IPsec traffic

4.2.1 The data

As it was the case for unencrypted network traffic, all applications were monitored in an isolated network namespace, so that the captured traces only contain packets from the targeted application.

The tool that was used for capturing is no longer Wireshark due to some difficulties that occur when trying to run Wireshark on the iMinds Virtual Wall. From now on, all traces are captured using the built-in tcpdump command. Skype traces were captured while running version 4.3.0.37, the torrent client that was used is Deluge 1.3.12 and Mozilla Firefox 52.0 served as browser for obtaining HTTP and YouTube traces. All traces were captured with caching disabled. Both physical machines on the iMinds Virtual Wall ran Ubuntu 16.04 LTS as operating system.

An overview of the traces used further in this chapter can be found in table 4.1. The numbers are calculated from 10 captured traces of each type.

A Skype trace consists of either calling a person, or being called, some time for the actual conversation, and ending the call. The YouTube traces contain packets from surfing to the YouTube homepage, and opening a random video that is watched until the end. The HTTP traces cover the Google search and opening one of the suggested webpages. And the BitTorrent traces start at the point of opening a torrent file in the torrent client, and goes on for a brief moment of downloading the actual content.

<table>
<thead>
<tr>
<th></th>
<th>Mean #packets</th>
<th>Max #packets</th>
<th>Min #packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skype</td>
<td>11251.3</td>
<td>19434</td>
<td>9681</td>
</tr>
<tr>
<td>HTTP</td>
<td>11734.1</td>
<td>57358</td>
<td>2342</td>
</tr>
<tr>
<td>YouTube</td>
<td>16274.8</td>
<td>26093</td>
<td>7379</td>
</tr>
<tr>
<td>BitTorrent</td>
<td>9982.5</td>
<td>9989</td>
<td>9978</td>
</tr>
</tbody>
</table>

4.2.2 Feature spaces

The same two feature spaces as in the previous chapter were extracted from the data. Visual inspection shows the similarities between the encrypted and the unencrypted data. However,
some differences occur as well. Some of these differences are caused by the change of environment. Another environment can cause a slight change in timing characteristics, although the global trend remains unchanged. Changes in the packet size on the other hand are caused by the additional headers and paddings that are part of the IPsec protocol.

The visualizations of both feature spaces are added in figures 4.3 and 4.4, so that the reader can compare them to the figures from the previous chapter.

Figure 4.3: The 2D-histograms (packet size and IAT) of all four IPsec encrypted traffic types.
4.2 Classification of IPsec traffic

(a) Feature space HTTP window
(b) Feature space Skype window

(c) Feature space Torrent window
(d) Feature space YouTube window

Figure 4.4: The 2D-histograms (burst time and size) of all four IPsec encrypted traffic types.

4.2.3 Classifiers

Since the feature spaces are similar, results comparable to those of the previous chapter were expected. The results for all four classifiers and for the three different inputs are summarized in table 4.2, and show indeed close resemblance to the results that were obtained for unencrypted traffic. The low accuracy for the MLP classifier with the burst feature space is caused by the limited hyperparameter tuning that has been performed.
4.2 Classification of IPsec traffic

Table 4.2: Comparison of different classifiers.

<table>
<thead>
<tr>
<th>Accuracy [%]</th>
<th>Naive Bayes</th>
<th>Logistic regression</th>
<th>Random forest</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>93.55</td>
<td>95.70</td>
<td>96.77</td>
<td>92.47</td>
</tr>
<tr>
<td>Burst features</td>
<td>87.10</td>
<td>94.62</td>
<td>93.55</td>
<td>38.71</td>
</tr>
<tr>
<td>Both</td>
<td>94.62</td>
<td>95.70</td>
<td>96.77</td>
<td>93.55</td>
</tr>
</tbody>
</table>

A remarkable but not really surprising fact is that the optimal values of the hyperparameters for the three best performing classifiers are identical in the encrypted as well as in the unencrypted case. The hyperparameters for the multi-layer perceptron slightly differ, but they are not optimal as stated before.

4.2.4 Analysis of the results

In order to understand the achieved results, we will discuss the operation of IPsec in more detail. Since the feature spaces are defined using timing and packet size information, we will focus on the effects of IPsec on these characteristics.

IPsec can operate with a lot of different parameters. We will only discuss the setting in which the experiment took place. As mentioned previously, we used IPsec in tunnel mode. To encrypt the data, we rely on the Encapsulating Security Payload (ESP) as security protocol. ESP offers support for various encryption algorithms. The one that we opted for is the Advanced Encryption Standard (AES), with a 128-bit key. This encryption is combined with a SHA-256 as the hash function to provide data integrity.

The delays caused by AES-128 and SHA-256 are neglectable, since both have very efficient implementations.

The effect on the packet size will be explained using figure 4.5.

When using tunnel mode, the payload data is the actual packet that is to be sent (see figure 4.2). The figure shows an optional padding field. By default, this padding is not enabled, so we chose not to use it. The effect of our IPsec configuration on the packet size is therefore limited to an additional IPv4 header (typically 20 bytes), the security parameters index (4 bytes), a sequence number (4 bytes), some padding to reach a multiple of 4 bytes for the payload length, and finally the integrity check value, which is a SHA-256 hash (32 bytes). The reasons why our
classifiers can cope with this, are both the fact that all packets are affected by approximately
the same addition of information, and the use of a logarithmically scaled packet size axis.

4.3 A non-ideal environment

So far, we have only considered the ideal situation in which all captured packets originate from
a single application. In a more realistic scenario, several background processes may be running
and sending/receiving packets as well. This section will discuss the effect of capturing these
additional packets on the classification accuracy.

4.3.1 Characteristics of background traffic

A first option to add background traffic to the traces is capturing the packets from all inter-
faces on the client instead of just the interface from which the IPsec tunnel starts. The rate of
the background traffic captured from this additional interface (the control interface) is roughly
50,000 packets/s. Since we assume that such a high packet rate would possibly predominate
the packets originating from the actual application, we looked into the characteristics of back-
ground traffic on a PC intended for typical daily use. The packet rate of the background traffic
in such case is about 20 packets/s. Another observation about the characteristics of the back-
ground traffic is the rather small packet sizes, being maximum 150 bytes. Although this kind
of background traffic is not available on the iMinds Virtual Wall, it can be simulated with a
packet generator. To mimic the characteristics as well possible, a script was used to generate
packets of random length, smaller than 150 bytes, at a desired packet rate between 0 and 400
4.3 A non-ideal environment

This results in a steady packet stream of packets with an average inter-arrival time of the inverse packet rate. Because of the randomness of these packets, the receiving end, the server node, does not reply to them, resulting in upstream background traffic only. To simulate a more realistic scenario in which there is both upstream and downstream background traffic, the packet generation script ran on both the server and the client node.

4.3.2 Effect of background traffic

In order to test the effect of background traffic on the classification accuracy, traces containing this background traffic were generated. The way this is done is very similar to what is described in subsection 4.2.1. There are two differences. The first being the fact that this time the packet generation script is running (either on the client node or on both nodes) as well as the applications. And the second differences is the use of Ekiga version 4.0.1, which is an alternative VoIP application, instead of Skype due to login troubles on the iMinds Virtual Wall.

A visual examination of the feature spaces already shows the effects of the added background traffic quite well. For both the lowest and the highest packet rate of background traffic, both feature spaces are shown in figure 4.6. The figure is limited to a single traffic type, being VoIP traffic, however the same effects are observed for all types of traffic. For the lower packet rate, not a lot of background traffic packets are included in a single window, and thus the observed effect is barely noticeable. For the higher packet rate, the feature spaces are more distorted, as windows tend to have a lot of background packets compared to the actual application packets.

(a) Packet size & IAT feature space with 20 packets/s up- and downstream background traffic

(b) Packet size & IAT feature space with 400 packets/s up- and downstream background traffic
4.3 A non-ideal environment

(c) Burst feature space with 20 packets/s up- and downstream background traffic

(d) Burst feature space with 400 packets/s up- and downstream background traffic

Figure 4.6: Both feature spaces for VoIP traffic with additional background traffic.

For every tested packet rate, the traces were generated and the logistic regression classifier was trained with the optimal parameters obtained via 4-fold cross validation. The results for the packet size and IAT feature space are summarized in figure 4.7. It is remarkable that for every packet rate, the optimal hyperparameters that were obtained are the same. This suggests that a single model is capable of handling different packet rates.

Figure 4.7: The effect of background traffic on the classification accuracy.
Although the global trend is obvious, some irregularities need some explaining. As expected, the classification accuracy drops as the packet rate increases. This is easily explained by the fact that for a higher packet rate, a window (representing a single sample) contains less application specific packets and more background traffic packets, thus complicating the classification process. The irregularities are due to the random nature of the traces, and are only a couple of percents. The human interaction involved in for example the generation of HTTP traces can have a big impact on the content of the windows, since this depends heavily on the speed wherewith the user clicks links. A smoother graph might also be obtained if the experiment was done multiple times and the average accuracy over these experiment was plotted instead of the accuracy of a single experiment. Because of the intensive data generation process, this has not been done.

4.4 Conclusion

We have captured IPsec-encrypted packets on the iMinds Virtual Wall. The IPsec configuration established a tunnel between the client and the server, and all packets going through this tunnel where encrypted using ESP. We have shown that the feature spaces from the previous chapter are still valid, and that accuracy measures comparable to those of the previous chapter have been achieved. A more in depth look into the used encryption algorithms explained the obtained results. In the second part of this chapter, we considered a non-ideal environment in which some artificial background traffic was created. As the packet rate of the background traffic increased, the accuracy dropped. However this drop in accuracy is very moderate if we consider that the actual packet rate of background traffic measured on a normal computer was about 20 packets/s.
Chapter 5

Multi-label classification of
IPsec-encrypted network traffic

Introduction

Up to this point, we have always assumed that only one of the four applications was running at each time instant. We have however considered the possibility of capturing some background traffic packets. A next step to make the experiment more comparable to a realistic scenario is the use of multi-label classification. This means that every sample can have one up to four labels, which is something that might occur in a realistic environment. A person can watch YouTube videos while downloading content via a Torrent client for example. This chapter will first discuss multi-label classification without background traffic, and the major differences compared to single-label classification that pop up in this case. And afterwards, the effect of background traffic is again studied.

5.1 The data

The applications used are the same as in subsection 4.2.1, except for the use of Ekiga instead of Skype, as stated earlier. This implies that these traces can be reused as part of the new dataset, with only a single label. To generate multi-labeled traces, we used a slightly different approach in order to label each window as correctly as possible. Since all packets are encrypted, we cannot distinguish packets from different applications with deep packet analysis (for example in
Wireshark). Therefore we label every window with the applications that were running at that time, although that specific window might not contain packets from all running applications. With this approximation, we can later on explain certain behavior. To make sure that most windows contain packets from all running applications, we only start the capturing after the applications have started, e.g. a VoIP-call is already ongoing or the Torrent client has already started downloading. This is different from the previous approach that captures also the initialization phase. We can justify this approach because of the resemblance between windows and full traces, as discussed in subsection 3.2.1. The final dataset contains 526 single-labeled, 642 dual-labeled, 428 triple-labeled and 107 quadruple-labeled samples.

5.2 Feature spaces

Since we are working with samples with one up to four labels, there are 15 different possible combinations ($2^4 - 1$). Due to the nature of the chosen feature spaces, being packet size & IAT or burst characteristics, the histograms for multi-labeled samples do not resemble the addition of their single-labeled components. Although packet sizes might show this behavior, inter-arrival times do not. They tend to shorten as packets from different applications interleave one another. Also, bursts from one application may be interrupted by packets from the other application.

The plots in figure 5.1 show the packet size & IAT feature space for all possible dual-labeled combinations. The components of the single application histograms are noticeable and interpretable in the combined histograms. For example the presence of VoIP-packets, labeled as Skype in order to keep a consistent labeling in the entire book, is visible because of its size component that occurs, and the downward shift of the complete histogram due to the smaller inter-arrival times because of the interleaving VoIP-packets. This is most noticeable when VoIP-packets are present, since these packets have small inter-arrival times by default, see figure 4.3b. The histograms of the burst characteristics for dual-labeled samples are shown in figure 5.2.
5.2 Feature spaces

Figure 5.1: The 2D-histograms (packet size and IAT) of all dual-labeled IPsec-encrypted traffic types combinations.
5.2 Feature spaces

(a) Feature space HTTP-Skype window  
(b) Feature space HTTP-Torrent window  
(c) Feature space HTTP-YouTube window  
(d) Feature space Skype-Torrent window  
(e) Feature space Skype-YouTube window  
(f) Feature space Torrent-YouTube window  

Figure 5.2: The 2D-histograms (burst characteristics) of all dual-labeled IPsec-encrypted traffic types combinations.
5.3 Classifiers

When performing multi-label classification, some adaptations have to be made, compared to the single-label case. Not all classifiers that we used before are capable of predicting multiple labels for a single sample. Most of them can however be wrapped in a One-Vs-Rest classifier (which they already inherently use for multi-class classification). This method consists in fitting one classifier per class. Each classifier gets trained to predict that a sample either corresponds to its class, or to the rest (the other classes) [17]. Since the random forest classifier possesses this ability by default for multi-labeled data, it will be the classifier of our choice for multi-label classification.

So far, when using 4-fold cross validation to find the best hyperparameters, we optimized the accuracy of the classifier. When dealing with multi-label classification, this metric might no longer be the desired one to optimize, because it does not take into account partially correct predicted samples, as we have explained in the introductory chapter.

For an optimal number of trees (within a certain search range) the metrics in table 5.1 were obtained for the different feature spaces.

Table 5.1: Comparison of classifier performance for the different feature spaces.

<table>
<thead>
<tr>
<th>Feature Space</th>
<th>(Subset) accuracy [%]</th>
<th>Hamming loss [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>83.87</td>
<td>4.62</td>
</tr>
<tr>
<td>Burst features</td>
<td>76.83</td>
<td>6.89</td>
</tr>
<tr>
<td>Both</td>
<td>83.58</td>
<td>4.62</td>
</tr>
</tbody>
</table>

While the numbers in table 5.1 give an indication of how well the classifier performs, there is no such thing as a confusion matrix to visualize some class-specific metrics. A way to make this more clear is to look at the classification report. The classification report for the packet size & IAT feature space can be found in table 5.2.

1An average Hamming loss of 4.62% means that, on average, 4.62% of the predicted labels are incorrect (both false negatives and false positives).
## 5.4 A non-ideal environment

When dealing with single-label classification, we introduced background traffic in order to simulate a more realistic environment. The same approach was taken to study the effect of background traffic on multi-label classification. For each packet rate, the entire dataset (one up to four simultaneous applications) was generated. Only the case in which both upstream and downstream packets are generated is considered here, since this scenario is closer to what we would expect in the real world. The graph in figure 5.3 shows both the accuracy and the Hamming loss metric for different packet rates.

In multi-label classification, the effect seems to be much larger compared to the single-label case. The irregularities in this graph are again explained by the human interaction when generating traces. This human interaction has even more effect in this case, since packets from different applications might be captured during the time that is needed for the human to interact. Of course this is behavior that is expected, for example when surfing the web, but it impacts the ability to label the samples correctly.

The graph also shows that the subset accuracy is a rather harsh metric when dealing with multi-label classification. The subset accuracy tells us that for a background packet rate of 800 packets/sec we obtain only 56.65% fully correct predicted samples. While the Hamming loss at the same rate is 13.45%, which means that on average 13.45% of the predictions is incorrect.

### Table 5.2: Classification report for the packet size & IAT feature space.

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>95.00</td>
<td>95.00</td>
<td>95.00</td>
<td>168</td>
</tr>
<tr>
<td>Skype</td>
<td>100.00</td>
<td>94.00</td>
<td>97.00</td>
<td>170</td>
</tr>
<tr>
<td>Torrent</td>
<td>99.00</td>
<td>98.00</td>
<td>99.00</td>
<td>174</td>
</tr>
<tr>
<td>YouTube</td>
<td>88.00</td>
<td>98.00</td>
<td>92.00</td>
<td>193</td>
</tr>
<tr>
<td>avg / total</td>
<td>95.00</td>
<td>96.00</td>
<td>96.00</td>
<td>705</td>
</tr>
</tbody>
</table>

As we discussed in section 5.1, these numbers are an estimate of the performance, since we are unable to verify the true labels of each window.
5.5 Conclusion

We extended the experiment from the previous chapter so that the data contained multi-labeled network traces. We observed that the effect on the feature spaces is not purely additive, because of the nature of the timing feature. We limited ourself to a single classifier, being the random forest classifier, because of its inherent multi-label support. We also introduced an additional performance metric, the Hamming loss, since it is more robust than the accuracy when dealing with multi-label classification. Also, the use of classification reports helped us gaining more insight in the results. Again, we studied the effect of background traffic, which seemed more severe in this case than in the single-label classification problem.

Figure 5.3: The effect of background traffic on the classification accuracy.
Chapter 6

Tor-encrypted network traffic analysis

Introduction

In the previous chapters, we have studied both unencrypted and IPsec-encrypted network traffic. While IPsec guarantees data confidentiality by encryption of the IP packets, it does not offer anonymity, unless it is combined with a VPN service, which we will not discuss. A way of ensuring anonymity over the Internet is by using Tor, which provides both encryption and anonymization, thus hiding the IP address of the user from the outside world. The method that Tor uses for this will be explained in section 6.1. In this chapter we will discuss the ability of our classifiers to cope with the more complex methodology of Tor.

6.1 The Onion Router

Before going into the Tor topology and its configuration on the Virtual Wall, we should discuss its methodology. To ensure anonymity, Tor routes all packets through a Tor circuit, which consists of at least (and by default) three Tor relay nodes. These relay nodes only know the preceding and the next hop in the circuit. To connect to any of these relay nodes, the Tor client selects one randomly from a list, which is provided by the Tor directory server. The client then instructs this relay node to extend the circuit, until the required amount of relay hops are part of the circuit. For improved anonymity, the circuit is changed approximately every 10 minutes.
Although the client only knows how to connect to the first relay node, it obtains a secure “tunnel” with each of the relay nodes on the circuit, for which the parameters are obtained during the extension process. The client encrypts the data as many times as there are relay nodes in its circuit. The first encryption is done with the last obtained parameters (those from the final relay node), then with those obtained before that, and so on. This way, each relay node can only “peel” away one layer of encryption, since it does not know the parameters with which the deeper layers are encrypted. This layered structure of encryption is where the Tor name comes from. Figure 6.1 shows, in a simplistic way, how a circuit is established, without taking into account the exchange of encryption parameters. In figure 6.2, the layered encryption scheme is visualized.

![How Tor Works: 1](image1)

(a) Choosing an entrance Tor relay node

![How Tor Works: 2](image2)

(b) Extending the Tor circuit

Figure 6.1: The establishment of a Tor circuit [7].

### 6.2 Experimental setup

As it was the case for our IPsec experiments, the Tor experiments were also performed on the iMinds Virtual Wall, because, once again, multiple nodes were needed for the configuration of a Tor setup. The remainder of this section discusses both the iMinds Virtual Wall setup, which handles the routing configuration, and the configuration of Tor itself.

The Tor experiment consists of three nodes. A client, to run the applications, which we call the workstation in order to be consistent with the terminology of the Tor topology that we set up, a gateway, which acts as a proxy for the workstation, and a router, which is the default router for the gateway.
6.2 Experimental setup

Packets coming from the router can reach the public IPv4 Internet because they are NAT’ed by the Virtual Wall router. On the router, IPv4-forwarding is enabled and it is able to make use of NAT to forward packets coming from the gateway. At first thought, the router node seems to be unnecessary. However, its presence allows us to capture packets on the link between the router and the gateway, which is a private link, instead of on the control interface of the gateway, on which a lot of additional Virtual Wall specific packets are exchanged, if it was directly connected to the Virtual Wall router. The configuration so far only offers public IPv4 Internet connectivity to the router and the gateway, since the gateway is configured to not forward any IPv4 packets because it should act as a proxy and not as a router.

The preceding section discussed the methodology by which Tor operates. Now we can discuss how we used it on the Virtual Wall. According to the official Tor website [7], the safest way to use Tor is by using the Tor browser bundle. Using this bundle ensures that all traffic originating from the browser is effectively routed through the Tor circuit. Since we do not only need browser traffic routed over Tor but also VoIP and Torrent traffic, another solution is needed. This solution is the isolated proxy concept, and is known to be perfectly secure, but it does only
provide Internet access to applications that are configurable to use a SOCKS-proxy. To set up an isolated proxy, a Tor client is run on the gateway node, and the gateway opens its SOCKS-port to act as a SOCKS-proxy. The workstation should have no other connections than the connection to the gateway. This is the case if we only consider IPv4 traffic on the workstation since all Virtual Wall control traffic is IPv6. By configuring the applications (Firefox, Skype and Deluge) to use the gateway as SOCKS-proxy for all traffic, we obtain Tor-encrypted traffic originating from the Gateway. This is why we capture packets on the link between the gateway and the router.

6.3 Classification of Tor traffic

6.3.1 The data

The way the data was gathered during this experiment differs only slightly from how it was done in the IPsec experiment. In this case, applications have no longer been run in an isolated namespace, since we capture on the Tor-encrypted link between the gateway and the router, which only contains packets to and from the SOCKS-proxy, which are the application packets.
6.3 Classification of Tor traffic

The tool that was used for capturing is the tcpdump command. Skype traces were captured while running version 4.3.0.37, the torrent client that was used is Deluge 1.3.12 and Mozilla Firefox 52.0 served as browser for obtaining HTTP and YouTube traces. All traces were captured with caching disabled. All physical machines on the iMinds Virtual Wall ran Ubuntu 16.04 LTS as operating system.

For each traffic type, 10 traces were captured. This time, each capturing process continued until the trace contained 10,000 packets.

A Skype trace consists of either calling a person, or being called and some time for the actual conversation. The YouTube traces contain packets from surfing to the YouTube homepage, and opening a random video. The HTTP traces cover the Google search and opening one of the suggested webpages. If the 10,000 packets were not captured after the page was loaded, a link on that page was clicked until the 10,000 had been captured. And the BitTorrent traces start at the point of opening a torrent file in the torrent client, and goes on for a brief moment of downloading the actual content.

6.3.2 Feature spaces

As in the previous chapters, the packet size and IAT feature space and the burst feature space were extracted from the data. They are both visualized in figures 6.4 and 6.5. These figures are comparable to the ones shown in previous chapters, but some differences are noticeable. First of all, in the packet size and IAT feature space, the histograms show that the packet characteristics are far more spread out when using Tor. In the IAT dimension this caused by the route (circuit) that is followed by the packets. Since the Tor relay nodes are shared by all Tor users, a lot of fluctuations in timing may occur. And also in the packet size dimension, this is most clear when comparing Skype traffic from the Tor experiment with that of the IPsec experiment, we can observe some differences. In normal circumstances, Skype operates over UDP. But when using Tor, in combination with a SOCKS-proxy, Skype traffic is sent over TCP, changing both timing and packet size characteristics.
6.3 Classification of Tor traffic

(a) Feature space HTTP window

(b) Feature space Skype window

(c) Feature space Torrent window

(d) Feature space YouTube window

Figure 6.4: The 2D-histograms (packet size and IAT) of all four Tor-encrypted traffic types.

In the burst feature space on the other hand, much less differences are observed, because Tor (and the SOCKS-proxy) does not directly interfere with the burstiness of the traffic.
6.3 Classification of Tor traffic

(a) Feature space HTTP window  
(b) Feature space Skype window  
(c) Feature space Torrent window  
(d) Feature space YouTube window

Figure 6.5: The 2D-histograms (burst time and size) of all four Tor-encrypted traffic types.

6.3.3 Classifiers

Although the feature spaces do show some differences compared to the previous chapters, we still expected decent results, since, based on the visual inspection, the feature spaces of the different types are still reasonably distinguishable. The results for all four classifiers and for the three different inputs are summarized in table 6.1.

Since the packet characteristics are more spread out, the classification is less accurate. In general, there is a drop of 10 to 15% accuracy. Except for the MLP classifier, which has not been properly tuned, and is therefore not very useful for comparison. A general observation is the robustness of the burst feature space, which suffers less of an accuracy drop.
6.3 Classification of Tor traffic

Table 6.1: Comparison of different classifiers.

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Naive Bayes</th>
<th>Logistic regression</th>
<th>Random forest</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; IAT</td>
<td>80.56</td>
<td>77.78</td>
<td>84.72</td>
<td>58.33</td>
</tr>
<tr>
<td>Burst features</td>
<td>86.11</td>
<td>80.56</td>
<td>86.11</td>
<td>56.94</td>
</tr>
<tr>
<td>Both</td>
<td>83.33</td>
<td>80.56</td>
<td>80.56</td>
<td>59.72</td>
</tr>
</tbody>
</table>

6.3.4 Analysis of the results

In the preceding sections, the basic principles of Tor have been explained. In order to fully understand the achieved results, this section will discuss the protocol in more depth. As we did for IPsec, we will focus on the effect Tor has on the packet size and timing characteristics of a network trace. The effect on the timing can be explained by considering two causes. The first cause is the longer route that is taken through the Tor circuit, as described in section 6.1. It should be intuitively clear that this way of routing packets introduces an additional delay, which stays roughly the same during the time that the circuit is used. Different circuits cause different delays. A second aspect of Tor that might have an effect on the timing is the encryption that is used. Tor uses Transport Layer Security version 1.2 (TLSv2) to secure the connections between the client and each relay node. TLSv2 offers a wide variety of encryption algorithms, message authentication codes (MAC) and key exchange procedures. The algorithm that will be used for the actual data exchange is decided on during the TLS handshake. Since we did not capture the handshakes that take place during the construction of the circuit, and because different relay nodes may support different encryption standards, we can’t be sure about the encryption standard that is used. However the Advanced Encryption Standard (AES) is the most popular one, and has the highest priority to be chosen if it is accepted by the two endpoints of the TLS connection. Although, we do not know the key length with which AES is used (there are two possibilities, 128 or 256 bits), we can neglect the effect of the encryption and decryption on the timing characteristics. Messages are—most often—authenticated with a SHA-1 MAC, for which the delay is also neglectable.

The effect of Tor on the packet sizes is caused by the TLS encryption by each of the relay nodes that make up the circuit. Every node adds an additional layer of TLS security. The structure of a TLS record is shown in figure 6.6. The content type, major version and minor version are all 1 byte long, and the compressed length is 2 bytes long. At the end of the payload, the 128 bit
SHA-1 MAC is appended. As the figure indicates, the payload might possibly be compressed, however in practice, this feature is never used in TLS. This TLS record format adds 21 additional bytes per relay node to each packet. Since Tor uses three relay nodes to construct a circuit, a packet—at the client—is 63 bytes longer than the unencrypted packet. This is comparable to the increase in packet size caused by IPsec. Therefore, the additional routing delay, introduced by the anonymizing aspect of Tor, is the most important cause of the reduced accuracy.

6.4 Conclusion

We took a similar approach as we did for IPsec, by capturing traces on the iMinds Virtual Wall. The Tor setup that we chose is called an isolated proxy. In this setup, the gateway acts as a SOCKS-proxy and it routes all traffic that originates from the workstation over the Tor network. The major difference with IPsec is that Tor has a bigger effect on the timing characteristics of a network trace. The impact of this is clear from the achieved accuracies, which reach about 80% to 85%. As for all machine learning problems, more data would increase the achieved accuracy, because the classifiers would be able to learn from traces originating from more different Tor circuits, so they would be able to handle this “random” delay as noise.
Appendix A

The trained logistic regression classifier

The logistic regression classifier for multi-class classification is trained using a one-vs-rest approach. This means that for every possible class, a classifier is trained by assuming that a sample either belongs to that class (the one), or to the other classes (the rest). In order to provide a better understanding of the trained classifiers, this appendix contains a visualization of the weights of each of the classifiers. High weights indicate that those features are likely to be discriminative for the class, while low weights suggest that these features are more likely to be high for the other classes.

The obtained weight matrix will be shown next to the feature space of a random window of the class for which the classifier was trained. This is done to provide optimal understanding of the weights to the reader.
A.1 Unencrypted network traffic

A.1.1 Packet size and inter-arrival time

Figure A.1: The weights of the classifier trained for the HTTP class, and a random HTTP window.

Figure A.2: The weights of the classifier trained for the Torrent class, and a random Torrent window.
A.1 Unencrypted network traffic

Figure A.3: The weights of the classifier trained for the Skype class, and a random Skype window.

Figure A.4: The weights of the classifier trained for the YouTube class, and a random YouTube window.
A.1.2 Burst features

Figure A.5: The weights of the classifier trained for the HTTP class, and a random HTTP window.

Figure A.6: The weights of the classifier trained for the Torrent class, and a random Torrent window.
A.1 Unencrypted network traffic

Figure A.7: The weights of the classifier trained for the Skype class, and a random Skype window.

Figure A.8: The weights of the classifier trained for the YouTube class, and a random YouTube window.
A.2 IPsec network traffic

A.2.1 Packet size and inter-arrival time

Figure A.9: The weights of the classifier trained for the HTTP class, and a random HTTP window.

Figure A.10: The weights of the classifier trained for the Torrent class, and a random Torrent window.
A.2 IPsec network traffic

Figure A.11: The weights of the classifier trained for the Skype class, and a random Skype window.

Figure A.12: The weights of the classifier trained for the YouTube class, and a random YouTube window.
A.2 IPsec network traffic

### A.2.2 Burst features

**Figure A.13:** The weights of the classifier trained for the HTTP class, and a random HTTP window.

**Figure A.14:** The weights of the classifier trained for the Torrent class, and a random Torrent window.
A.2 IPsec network traffic

Figure A.15: The weights of the classifier trained for the Skype class, and a random Skype window.

Figure A.16: The weights of the classifier trained for the YouTube class, and a random YouTube window.
A.3 Tor network traffic

A.3.1 Packet size and inter-arrival time

Figure A.17: The weights of the classifier trained for the HTTP class, and a random HTTP window.

Figure A.18: The weights of the classifier trained for the Torrent class, and a random Torrent window.
A.3 Tor network traffic

Figure A.19: The weights of the classifier trained for the Skype class, and a random Skype window.

Figure A.20: The weights of the classifier trained for the YouTube class, and a random YouTube window.
A.3.2 Burst features

Figure A.21: The weights of the classifier trained for the HTTP class, and a random HTTP window.

Figure A.22: The weights of the classifier trained for the Torrent class, and a random Torrent window.
A.3 Tor network traffic

Figure A.23: The weights of the classifier trained for the Skype class, and a random Skype window.

Figure A.24: The weights of the classifier trained for the YouTube class, and a random YouTube window.
References


