VPN Fingerprinting

Network protocol detection inside virtual private network tunnels

R.S. Graafmans

Delft University of Technology

VPN Fingerprinting

Network protocol detection inside virtual private network tunnels

by

R.S. Graafmans

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on December 15th, 2021 at 15:00 hour.

Student number:	4299442	
Project duration:	January, 2020 – December, 2021	
Thesis committee:	Dr. Z. Erkin,	TU Delft
	Dr. R.R. Venkatesha Prasad,	TU Delft
	Dr. C. Doerr,	TU Delft, supervisor
	Dr. P. Zimmermann,	TU Delft

An electronic version of this thesis is available at `https://repository.tudelft.nl/`.

TU Delft

Abstract

Virtual private networks are often used to secure communication between two hosts and preserve privacy by tunneling all traffic over a single encrypted channel. Previous work has already shown that metadata of different secured channels can be used to fingerprint various kinds of information. In this work, we will dive into the encrypted tunnels as used by VPNs. We have collected automatically generated data of 9 network protocols sent over 8 different VPN solutions with 3 different rates for mixed traffic each. Due to the single combined traffic channel of the VPN, this work had to focus on packet-wise features instead of stream-wise ones, requiring the development of new features compared to related work. Both Random Forest and Markov Chains are trained to distinguish the network protocols by finding the patterns of the protocols in the developed features. We show that it is possible to fingerprint network protocols in all different scenarios based on the metadata available. Moreover, it was found that size features are more important than timing-related ones, especially when padding comes into place. Lastly, we show that obfuscations methods focussing on distorting size or timing patterns solely are not effective enough and future obfuscation methods should incorporate both features.

Preface

The realization of this thesis has been a journey of ups and downs. After only 2 months, the Covid-19 pandemic brought a change in the work atmosphere, which led to it that a large part of this work is done from behind my desk at home. Something that did not make it easy for research, motivation, but especially writing. Fortunately, I could always count on the support of friends and family, the warm welcome at both KPN CISO and Security Labs, as well as the continuous guidance of prof. dr. Christian Doerr, dr. Phill Zimmerman and ir. Harm Griffioen. It was a rollercoaster ride, but something to remember forever and it allows me to proudly present the work that lies in front of you.

R.S. Graafmans
Delft, December 2021

List of Figures

List of Tables

Contents

1 Introduction

The Internet has become increasingly important in our daily lives. In the last 15 years, the number of people that have access to the internet increased from 17% of the world population to 51% [5, 6]. In addition to a greater reach, people are getting more dependent on the internet in their daily lives. Accessing information, banking, getting an insurance quote, fixing a delivery, etc. are all done by using the internet nowadays. And as dependence grows, so has the call for better security in recent years.

One of the answers to this is HTTPS for example. The secure website traffic protocol was founded in 1994 and formalized in 2000 [7]. Eight years ago only 50% of the internet sites used HTTPS, which has now grown to approximately 95% of all sites indexed by Google [8]. By then, having a site without a secure connection was not limited to smaller websites only, but larger companies like Facebook as well. Facebook introduced the usage of HTTPS on their platform only in 2011 (seven years after the company started) and took another 2.5 years before they made it their default used protocol when visiting their website.

Yet, we should question ourselves what the current adoption rate would be, as the adoption rate of HTTPS might be influenced by multiple factors of external stimuli. First of all, Google changed its PageRank algorithm to lower unsecured websites in the search results which led to lower traffic numbers, often causing a loss of revenue from sales or advertisements. Secondly, LetsEncrypt's launch of free certificates made it possible for smaller websites to secure their website more cheaply, where paid certificates often do not outweigh the costs. Lastly, the new European General Data Protection Regulation (GDPR) obliges companies to protect their customer data sufficiently by handing out fines (where HTTPS can be seen as a basic security measure to prevent eavesdropping) [9]. Therefore, it seems that companies need to have a strong incentive, created by customers, large initiatives started by companies or law, before adopting new security measures or techniques.

One of the largest leaks of government secrets occurred in 2013 involving the National Security Agency (NSA), an intelligence agency of the United States of America. Edward Snowden, a former NSA employee, leaked highly classified information to various newspapers about various NSA programs and operations, exposing an unprecedented level of mass surveillance. As many people were shocked and even politicians requested answers from the NSA, they released a document in which they stated that they only touched 1.6% of the total internet traffic. By then this was around 29 petabytes a day of which approximately 7.5 terabytes were selected for review [10]. However, their claims did not go into full detail. It could be the case that the 1.6% was captured after some specific filtering was done. Assuming they filtered for network packets that have URLs embedded, (as in DNS and HTTP GET requests and which have a maximum size of 0.5 and 8 kilobytes respectively), the NSA could have collected more than 1×10^{12} URL requests per day at that time. Far more technological advancements have been made since then leading to cheaper data storage, bandwidth, and computing power allowing governments to increase their massive surveillance programs and it is expected to continue for some time under Moore's law [11].

As the leaks exposed an unseen level of privacy infringements, it could be one of the factors that caused the rise of privacy protection usage. Such a tool is a Virtual Private Network (VPN), which tunnels all your connection between your device and a trusted server before it is sent to its final destination. As these servers are used by multiple people it is difficult to track the origin of the network traffic making it for agencies to perform mass surveillance. According to market research [12], the usage of personal VPNs by American citizens has grown from 0.13% in 2010 to 4.65% in 2019. Although this is still a relatively small number of users on a day-to-day basis, people seem to get more aware of risks and privacy issues on the internet. However, VPNs are not only used for privacy reasons as criminals are often using VPNs to hide their location and preserve tracking of their other activities.

1.1. Fingerprinting

As said, encryption techniques can be used to hide the content of communications in such a way that it cannot be read directly. This does not mean that nothing can be learned by observing the encrypted traffic as patterns may be present. For example, when it would be snowing outside, looking at rooftops could tell us something about the temperature inside the houses. When there is a clear spot between snowy roofs, it could indicate a possible cannabis farm. So without knowing about what is happening inside the house, one could still draw some possible conclusion about it. On the other hand, one could try to invent some countermeasures to prevent fingerprinting. In the aforementioned example, better thermal isolation could be a solution for the ability to fingerprint a cannabis farm.

This example shows the basic idea behind fingerprinting. This principle has been studied for a relatively long time now, with papers from 2003 already researching the fingerprinting of webpages [13]. Webpages were quite static, meaning that the content did not change as often as it does nowadays. This property made it possible to conclude which web pages were requested by observing the different size sequences of received packets. Over the years the technique became less successful since webpages and their content changed more frequently, and the number total of websites grew.

Fingerprinting is not limited to content only, as patterns of applications, malware, network protocols, or operating systems might be observed as well. As long as the observed data contains enough information, it should be possible to create decision models that could automatically distinguish the different items. A decision model can be quite simple when there is a specific property that can be used for fingerprinting. A great example of this was the Cobalt Strike detection model, which is a frequently used hacker tool, where one single anomalous whitespace in the traffic headers gave away the usage of this tool [14]. However, it can be more challenging as well, as shown by the android application fingerprinting model which required the usage of more advanced machine learnings models [15].

Researching fingerprintability can be useful for different parties. When the detection of certain information is unwanted, for example, the usage of VPNs in countries where freedom of the press cannot be taken for granted, it is necessary to limit the unique patterns that can be observed. On the other hand, finding malware patterns allows virus scanners to better detect infections, preventing cybercriminals from doing their business.

1.2. Virtual private networks

As mentioned before, the usage of VPNs is rising. More people tend to use it to protect their privacy, secure their data, or bypass geographically blocked content. Where the number of public Wi-Fi hotspots is growing on one hand and the availability of services on mobile devices, on the other hand, we often tend to forget about the potential risks of dealing with important data of potential harmful connections. An example of such an operation is logging into your bank account. You want to make sure that your data (and money) is safe and no evil actions should be done to it. It is therefore that not only banks are warning their customers to avoid untrusted networks nowadays, but governments and (international) police departments do so as well. by making use of a VPN, every data connection is tunneled, meaning it is sent encrypted to a trusted server before being sent to the destination server. This adds an additional layer of security for all traffic between your device and the VPN server which makes it is harder to intercept and/or manipulate traffic.

Besides the security issues, privacy is of concern as well. Since 2013, RSF's (Reporters Without Borders) indicator for press freedom has dropped by 12%. Especially the Covid-19 situation seems to amplify the many crises which threaten the right to freely report [16]. The same organization (as well as many others) is stimulating the press to use VPNs to protect itself and ensure the free press. But one should not forget about the main use case of VPNs. Especially during the Covid-19 pandemic, remote-access VPNs are used to connect to work environments from different locations. This way local services could be accessed, company firewalls could monitor the traffic, and it helps to apply the company's (security) policies or updates.

1.3. Research goals

Previous research (which will be discussed later) shows that fingerprinting of VPN traffic is possible. However, the conducted research was very limited in terms of combined traffic situations, VPN protocols, and a deeper understanding of which aspects make fingerprinting possible. In this thesis, those gaps will be investigated. The goal is to find a better understanding of the possibilities and workings of VPN fingerprinting. This should help us to answer the following research question:

> *To which extent do VPN-protocols, -solutions and -obfuscations differ when comparing the fingerprintability of different network protocols used inside a VPN-tunnel?*

To answer this research question completely, we must look at multiple and different aspects. The following sub-questions can help to do this thoroughly and answer all the aspects of the main research question:

- Which data features are most important for the fingerprintability of VPN traffic?
- Which effect does the mixing of protocols have over an encrypted stream on the scoring metrics?
- How do different VPN protocols and solutions compare in fingerprintability?
- To which extent are some network protocols easier to fingerprint than others?
- Which mitigations can be taken to protect our data better and how do they good do they perform?

This research contributes to a better understanding of the security and privacy risks that come with using VPNs. This is done by 1) finding out which data features makes fingerprinting possible in the first place; 2) which VPN protocols are more vulnerable for this type of attack; 3) which network protocols are more easily detected; 4) in which direction measures should be designed to minimize potential fingerprinting as much as possible.

1.4. Readers guide

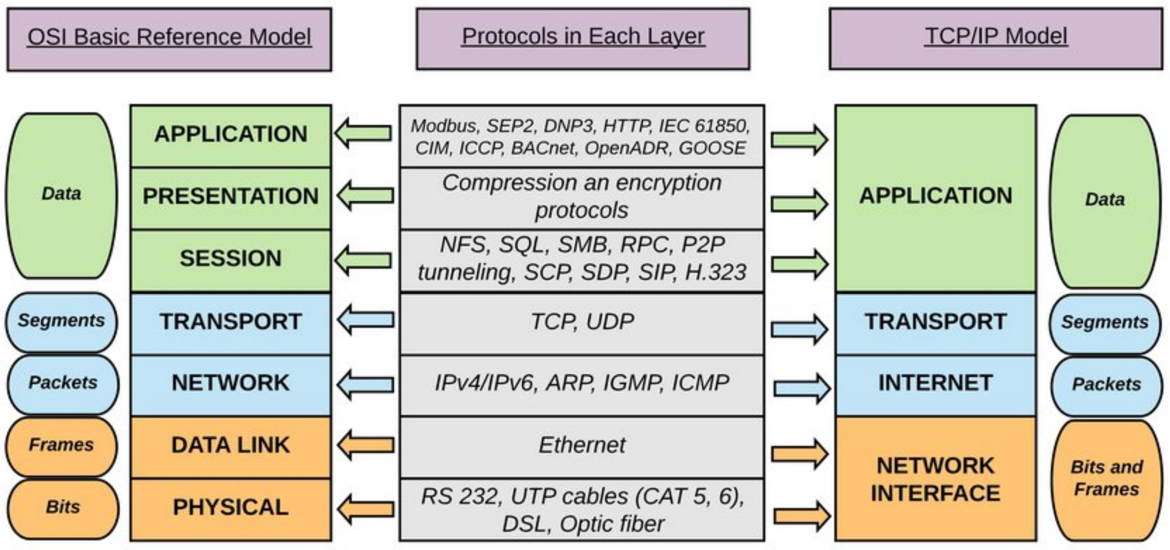
Before answering the questions, this report will first discuss the required background information. These will be explained in chapter 2. With this information and the understanding of it, related research will be discussed in chapter 3. Then the setup of this research will be explained in chapter 4, followed by the results in chapter 5. After discussing all the different sub-questions, we can conclude on the main question in chapter 6.

2 Background

In this chapter, some basic concepts of networks and their protocols will be explained. Thereafter we will look at some ways how to protect one's privacy better and some basic ideas of attacks that still try to extract information without attacking security itself.

2.1. Basic network concepts

To deal with different kinds of traffic types, the internet must have an underlying structure. Various applications can have very different purposes, but they have a lot in common in terms of data transfer needs. For this, a global structure has been designed, the so-called OSI and TCP / IP models (see figure 2.1). Often the TCP/IP model is described as the more practical implementation of the theoretical OSI model, which is describing the ideal situation.



OSI Basic Reference Model
Protocols in Each Layer
TCP/IP Model
Data
APPLICATION
PRESENTATION
SESSION
Segments
TRANSPORT
Packets
NETWORK
Frames
DATA LINK
Bits
PHYSICAL
Modbus, SEP2, DNP3, HTTP, IEC 61850, CIM, ICCP, BACnet, OpenADR, GOOSE
Compression an encryption protocols
NFS, SQL, SMB, RPC, P2P tunneling, SCP, SDP, SIP, H.323
TCP, UDP
IPv4/IPv6, ARP, IGMP, ICMP
Ethernet
RS 232, UTP cables (CAT 5, 6), DSL, Optic fiber
APPLICATION
TRANSPORT
INTERNET
NETWORK INTERFACE
Data
Segments
Packets
Bits and Frames

Figure 2.1: OSI - TCP/IP overview. [1]
Note: Although the model may suggest so, network packet fingerprinting is not limited exclusively to the internet layer. However, because of VPN characteristics which are explained later, this thesis will focus on fingerprinting via the internet layer.

Since the data we want to transfer can have all different sizes, it might cause performance issues when it is sent without splitting them into smaller blocks. Due to broken links, busy intermediate hops, interference, etc, some data might be delayed or not even delivered at all. By splitting the data into multiple (smaller) packets, different data sizes can be transferred in a more standardized way where maximum sizes are known in advance. This makes it possible to resend only parts of the data when it is discovered that a packet was not received correctly, for example, due to an intermediate hop that has disappeared.

Depending on the requirements of the application and traffic, the packet building process could start in different upper layers of the TCP/IP model. Each of them serves a different purpose to make data transfer possible. Figure 2.2 shows the full process of building packets, transmitting them over intermediate hops, and parsing them at the final destination. Although the TCP/IP model is combining the lowest 2 layers of the OSI model and referring to them as the network interface layer, they both serve different purposes for which they are split in figure 2.2.

Packets often need to travel through multiple physical connections to reach their destination. The internet layer is responsible for storing information about the final source and destination while the data link and physical layer handle the route between two physically connected devices. This is shown in figure 2.2 by different header colors at the intermediate hop. Moreover, every protocol adds its header (and sometimes tail) as an encapsulation around the package received from the layer directly above. Packets are always assembled starting from the top layer downwards and disassembled from the bottom layer upwards. However, it is not a requirement to include all layers, since packets could be build by starting from a lower layer. However intermediate layers cannot be skipped.

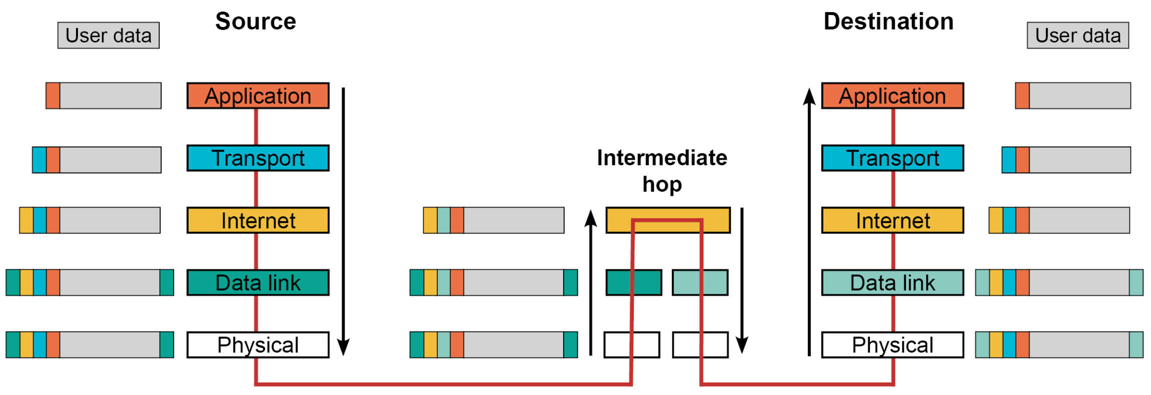


Figure 2.2: Packet building and transmitting order

2.2. Network Protocols

With a general structure in place to build packets and transmit them, the question raises how different tasks are fulfilled, like browsing the web, sending an email, or basic file transmissions. Each task has different requirements leading to the introduction of different network protocols, each responsible for its task. Network protocols are using the general building blocks and extend them with custom information, data schemes, and behavioral rules on top to meet the specific requirements. Below a few fundamental protocols that serve different basic user tasks will be discussed.

2.2.1. DNS

The Domain Name System (DNS) is a network architecture that can translate domain names towards the correct IP address. Using a hierarchical topology, DNS servers rely on other DNS servers as well. Luckily this last part is often not of concern for the end-user, since they only talk to the first server. The DNS protocol supports different types of records, all serving different purposes. They could differ from being an IP address connected to a domain name, specifying mail servers, or text requests used for verification purposes for example. Different types of security could be added to the DNS protocol. For example, DNSSec is used for the verification of answers to overcome spoofing, whereas DoH and DoT are used to encrypt the payload.

2.2.2. FTP

The File Transfer Protocol (FTP) is one of the first protocols which made it possible to transfer files between machines over the internet. It is optimized for file transferring including possibilities to list remote directories and transfer files over multiple channels at the same time. FTP is based on commands which can be sent from a terminal or being used in the background by modern graphical user interfaces. FTP is not encrypted by itself, therefor FTPS (FTP SSL) can be used. The other secure file transfer setup is SFTP (SSH FTP). Although the name sounds similar the implementation is quite different as it uses the SSH protocol as a basis instead of encrypting the FTP protocol.

2.2.3. HTTP

The Hypertext Transfer Protocol (HTTP) has grown over the years as the usage of the internet has increased. HTTP is mainly used for serving webpages and related files. It can handle various types of requests, from data requests to data modification or deletion requests. HTTP can handle a lot of different custom requests since the header can be extended by a lot of (custom) fields, of which each serves a different purpose. Although HTTP can be used to transfer files as well, it supports it only in a limited way and only those files that are exposed on the webserver. The original HTTP connection is unencrypted but can be extended with TLS (a widely used encryption standard) creating HTTPS (HTTP Secure).

2.2.4. ICMP

The Internet Control Message Protocol (ICMP) is used for supportive tasks. Although most people will know this protocol for checking if a host is alive or to trace the route a packet takes to from its source to its destination, it is also used to inform sending parties that some packets did not arrive correctly. The diagnostic process of checking if hosts are alive is also known as pinging.

2.2.5. IMAP

The Internet Message Access Protocol (IMAP) is used for the management of an email box. In comparison to POP, modifications to messages or locations are synchronized back to the server and stored. Most email applications are using this protocol nowadays. In the background, it uses predefined commands to handle the tasks.

2.2.6. POP3

The Post Office Protocol (POP) is another mail protocol for retrieving emails from a mail server. In comparison to IMAP, this protocol does not support synchronization across devices. POP uses commands in the background as well as many other basic protocols. Since POP is used only to retrieve mails, the size of the data stream could say something about the number of emails received (when by dividing by the average email size).

2.2.7. SIP-RTP

Voice over IP (VoIP) calls are based on two protocols. First of all the Session Initiation Protocol (SIP), which is used to set up, maintain, and terminating real-time sessions including voice and video. After a session is set up, the Realtime Transport Protocol (RTP) is used to deliver audio and video over the internet. Different codecs can be used to encode the payload, based on the needs (like quality and bandwidth) of the user.

2.2.8. SMTP

The Simple Mail Transfer Protocol (SMTP) is used to deliver emails to the correct servers. Just like POP and IMAP, a client often needs to identify and authenticate itself before being allowed to send other commands. However, normal users won't observe these handlings as these are all done in the background. Since SMTP is used only to send mails, the size of the data stream could say something about the number of mails send (when by dividing by the average email size).

2.2.9. SSH

The Secure Shell protocol (SSH) could be used for different purposes since a lot of network services could be secured with it. However, most user interaction with this protocol is based on its ability to set up secure connections to remote computers and execute different tasks via a command-line interface on them.

2.3. Anonymity tools for network traffic

Many of the above protocols do not contain encryption on their own. In the past 20 years, however, a lot of extensions are built for them introducing variants like HTTPS, SFTP, DoT, or simply extending the command options of POP or SMTP with STARTTLS (which sets up an insecure connection but allows to upgrade this.)

However, these encryption settings only encrypt the highest protocol payload (eg For HTTPS, the HTTP payload is encrypted, TCP is not). This way, IP addresses, and port numbers can be still read. Moreover, the same applies to plaintext information sent during the encryption setup. There exist different solutions to prevent this from happening. Most of them are doing this by encapsulating all the payload and encrypt it after which it will be routed via a custom route.

Depending on the needs of the user or company, different solutions can be chosen. Moreover, combining options is not uncommon in enterprise settings. The most frequently used options are VPN, Proxy, SSH tunnels, and anonymity networks. Although the scope of this research is limited to VPNs, other anonymity tools share important principles in general, which might make it possible to apply the key concepts of this research to the other anonymity tools as well.

2.3.1. VPN

Virtual private networks are one of the most secure options for hiding and protecting your network usage. The user initiates a connection to the VPN server, after which an encrypted tunnel is set up. All traffic is then routed through this tunnel, not limited to websites but all network protocols included. The VPN server routes the traffic coming from the tunnel to the correct destination. When secure protocols are used inside the tunnel, it is hard to correlate VPN server traffic back to the actual end-user. Since VPNs are the main focus of this research, the next section will further dive into the different VPN styles and protocols.

2.3.2. Proxy

Proxies are servers that can relay website requests to web servers. Instead of visiting a website directly, you connect to a proxy server, which then makes a connection towards the requested site. Although this could help to prevent the webserver from logging your IP, visiting geo-blocked content, and bypass some firewalls, one should be aware that some proxies still add some information in the HTTP headers which could identify you ('X_forwarded_for' or 'referrer' tags). Apart from the fact that most web proxies are free to use, it is questionable how they make money. Moreover, proxies often are limited to website traffic only. The biggest difference compared to VPNs is the fact that for every data stream a new connection has to be set up.

2.3.3. SSH tunnel

SSH tunnels are somewhat similar to VPN servers and proxies. A secure tunnel is built, but instead of tunneling all network traffic, only the packets sent to a specific port are being tunneled. This setup is often used to securely connect to a specific service or file share from outside the network. SSH tunneling does rely on the SSH protocol and implementations like SOCKS. Since SSH tunnels are often application-specific and need more extensive configurations, VPNs are often picked when one wants to make sure that all traffic would be encrypted.

2.3.4. Anonymity networks

Anonymity networks are, other than many other solutions, based on communities and peers. This is in contrast to other anonymity tools where everything depends on (paid) servers of specific companies. Although different anonymity networks exist like I2P and Freenet, The Onion Routing network (TOR) is one of the best-known and most used anonymity tool. It gets its name from the working of the service. By building a chain of relay servers, with an additional layer of encryption for each of them, it is hard to link users to their contents. By removing the single point of trust, it is often considered the safest option to maintain privacy and anonymity on the web. Besides access to the normal web, there are special tor web servers as well, which could only be visited using the TOR browser. This part of the internet is the so-called dark web.

2.4. VPN

To understand the differences between VPNs, we have to take a look at the different purposes as well as protocols and configurations. To start with, we can differ three main types of VPN usage:

- Consumer VPNs
- Remote access VPNs
- Site-to-site VPNs

Consumer VPNs serve a different purpose than remote access and site-to-site VPNs. Where Consumer VPNs are focussing more on bypassing (geo-) block and increasing privacy, remote access and site-to-site VPNs are mostly used to securely connect users and parts of networks. However, looking at the underlying setup, it could be said that consumer and remote access VPNs are closer to each other than site-to-site VPNs are. The latter is responsible to interconnect 2 networks, making it possible to let 2 machines from different networks talk to each other but not per-definition interacting with all other traffic. Consumer and remote-access VPNs however are forwarding all traffic from the host machine over a tunnel to the other side, often isolated from other users.

2.4.1. Tunnel vs Tap

There are two main subtypes of how VPNs could route traffic. When looking at the network stack, TAP VPNs operate on the data link layer (number 2) where ethernet frames are being handled. Tunneled VPNs operate on the network layer (number 3) where the IP protocol is being handled among others. Setting up site-to-site VPNs one normally wants to combine 2 different (remote) networks and make them behave like one. In such cases, it is important to allow packets to freely flow between all endpoints. In such cases, TAP networks are being used. This way all layer 3 protocols, (so not limited to IP,) can be used as well. This is an important feature to route packets correctly to the right (MAC) address for example.

When remote access to only a few machines or subnets, including a more strict security design, is desired, TUN-networks are often the way to go. This VPN type tunnels every IP packet between two hosts. The receiving host then routes the packets according to their configured routes and security policies. In this situation, the sender cannot control how the receiver will route the packet when it is received. Since this method is widely used nowadays, VPNs are often build-in by default enterprise firewalls nowadays. Policies then decide which traffic is accepted or declined, which can help to sanitize user's traffic. Since consumers only want to securely connect to the internet via VPN servers, the TUN setup is picked most of the time.

2.4.2. Protocols

During the last 25 years, different VPN protocols have been invented, used, and discarded. To limit ourselves a bit, only the protocols which are commonly used nowadays will be discussed.

OpenVPN
OpenVPN is one of the largest open-source VPNs and has been around for almost 20 years. It has its protocol and has a dedicated port assigned. Security-wise it supports most encryption algorithms provided in the OpenSSL library. The session is built by a custom security protocol based on TLS. It supports both TCP and UDP, letting users pick the protocol (and implied benefits and drawbacks) which fit them best. Since this protocol is widely supported and open-source, we often see this protocol being implemented in consumer routers as well.

OpenVPN supports the User Datagram Protocol (UDP) as well as the Transmission Control Protocol (TCP). Both protocols are built upon the IP layer. Where TCP (as the name suggests) implements all kinds of error-checking and recovering, back-and-forth communication, and deliverability, UDP skips those additions in a try to lower the overhead and delays. For OpenVPN connections, there is an important caveat, however. Since TCP connections come with a lot of overhead, a special algorithm is used by default. This so-called Nagle's algorithm buffer packets for a limited time and is then sending them combined. However, this could be of a negative influence when timing or size features are important.

IPSec
IPSec which stands for IP Security is a standard suite of protocols used to provide secure communication between machines. However, IPSec is always used alongside another protocol like IKEv2 for example. Those additions are used to set up a tunnel by providing identification and authentication and negotiating which encryption and authentication options are chosen for negotiating the tunnel parameters. The IPSec suite contains different protocols (serving different purposes) but when wanting to send encrypted data, the Encapsulating Security Payload (ESP) protocol is used.

Wireguard
Wireguard is a relatively new open-source VPN protocol. It is light weighted (approximately 1% of the number of lines of OpenVPN), fast, and easy to configure. Based on those aspects, people often expect Wireguard to replace OpenVPN over time to be the default opensource VPN protocol. Since March 2020, Wireguard is added to the kernel of Linux, and some of the biggest consumer VPN solutions now support Wireguard as well. However, as every machine should be configured individually, additional layers or management solutions should be made to make it more convenient to use for the less skilled end-user.

2.4.3. SSL-VPN
Although it isn't a VPN protocol on its own, SSL-VPN is widely used for corporate remote access VPNs nowadays. Another misconception from this name is SSL. Since SSL is not considered secure nowadays, TLS (its successor) is used to secure traffic. SSL-VPN can be used to securely visit an (internal) website via the browser without installing something (clientless) or it can set up a tunnel via specific client software and route all traffic through this tunnel. Especially this last one is often used.

A large advantage of SSL-VPN is the easiness of configuration and its usage. Since this setup is not usable for site-to-site VPNs, a lot of difficult configuration options can be skipped. The fact that it uses port 443 for its gateway helps to overcome limiting firewall policies when trying to connect to one. Since normal web traffic uses port 443 as well, a VPN tunnel cannot be distinguished from normal traffic since both are encrypted. With the introduction of DTLS (Datagram Transport Layer Security) the advantages of connectionless communication are combined with the easiness of this VPN solution. The only downside is the fact it uses a UDP instead of TCP, which makes it easier to block these connections.

2.5. Network fingerprinting
With all encryption options for different protocols and extensive anonymity tools in place, it seems logical to think that one's privacy is almost impossible to violate. By strictly looking at the network packets of VPN traffic it is indeed hard to tell something about the content of the network steams. However, anonymity tools and encryption techniques only take into account the security of each packet separately. Looking at the bigger picture, timing and size

patterns may become visible. A technique that uses these patterns to say something about these streams is called fingerprinting.

Network traffic fingerprinting is not a term with a strict definition. It can be performed based on signatures found in packets or statistical features from packets, bursts, or whole streams for example. Network packet signatures are used frequently for network intrusion detection systems. However, when packets and their headers are encrypted, it is often not possible to find such signatures to say something about the content. Therefore, it is almost certain that work related to our fingerprinting goals will be based on one of the following two methods below to classify network traffic:

2.5.1. Stream-wise

"A data stream is a countably infinite sequence of elements and is used to represent data elements that are made available over time." [17] This general definition of a stream applies to network connections as well. Each connection which transfers data between a source and destination is considered a network stream. Communication between two endpoints is not limited to one concurrent stream at a time. Especially when multiple applications talk to each other, multiple streams can live alongside.

However, when those streams can be separated from each other, they can be summarized via different statistical features. One could think of features like average size, standard deviation, median, number of packets, every xth percentile, etc. When accounting for both directions as well, one could imagine how rapidly the feature set grows with numbers that describe a stream.

2.5.2. Packet-wise

When a stream-wise analysis is not possible or wanted, another method that can be used is packet-wise fingerprinting. Without the requirement to have perfectly separated nor finished streams, this method can be applied on combined encrypted channels like VPNs as those prevent from splitting packets to separate streams. The features are based on a packet itself and its direct neighbors. Possible features are, for example, packet size, directional time deltas, number of consecutive packets in 1 direction, or delta size towards the last opposite packet.

3
Related Work

During the last decades, a lot of research has been conducted related to traffic analysis. During a continuous cat and mice game, new technologies are developed to protect the users' privacy better on one side, whereas research is becoming more advanced as well trying to find information leakage models. Over the years privacy evading techniques changed from counting file sizes of retrieved files to determine which website has been visited towards creating hashes to track adversarial systems or differentiating mobile application usage by their streams.

3.1. Fingerprinting categories

The most important part of this research is filling the possible gap between the different traffic analysis categories. The following subsections show all types of information that were determined. A further distinction between anonymity tools, machine learning techniques, and other factors is explained and compared per category.

3.1.1. Content

One of the earliest papers about fingerprinting web traffic is from Hintz [13]. Here the leakage of the so-called proxy 'Safeweb' is analyzed. By then, there was limited knowledge and interest in studying possible ways to eavesdrop on internet traffic. With this research, they tried to not focus on the possible attack vector alone, but some possible mitigations as well. As a lot of the websites didn't have a lot of changing content by then, the sizes of files being served on a webpage didn't change that often as well. This exact property was used in this research. By making fingerprints of every website, it was possible (on a small scale) to determine which websites were visited, based on the number of files and file sizes. The whole idea of using an encrypted proxy was broken by this. Because modern sites often use multiple frameworks for site functionality and the content changes faster, the attack is much more difficult to execute. Nevertheless, the main reason lies in the fact that current anonymity tools also hide the number of files and their size, making it impossible to perform this fingerprinting technique.

Another example of content fingerprinting was delivered by Bissias et al. [18]. Where previous research is using vulnerabilities in SSH or SSL, (for example by using the possibility to detect original file sizes,) this research tends to find a more general approach that will also work for VPNs, SSH tunneling and wireless networks. By generating traces for both packet sizes as well as inter-arrival times, profiles were created for different websites. The traces observed when wanting to classify a website can then be compared to earlier captured traces and by calculating the cross-similarity between all original profiles and the one to classify, it is possible to determine which website was visited. Multiple interesting results were found. First of all this attack worked with an accuracy of 23% for the first guess. However, pre-determining which websites are easier to identify could increase the accuracy towards 40% for the first guess and even 100% when giving it 3 chances. Secondly, using both size and time traces only improved the accuracy scores minimally compared to size-only traces. Using only the time traces, the accuracy scores were more than halved. Lastly, it was observed that including a more than 24-hour delay between creating fingerprints and classifying websites lowered the accuracy scores by a few percentage points. Although this research was also focussing on content fingerprinting, it showed a larger attack frame by moving away from a known vulnerability, since the features changed from file sizes to packet sizes. However, this research was conducted in 2006 when websites seemed to be more static and having fewer dependencies, which makes it questionable how good the results would be if repeated today. More importantly, the attack was more generalized by having features that are available to VPNs, but it did not address the other important part of anonymity tools, namely the possibility of having to deal with combined channels which makes it harder to create profiles since multiple streams cannot be separated from each other.

3.1.2. Protocols

A different category of fingerprinting is based not on the content itself but focuses on the protocols responsible for different data transfers. One of these studies was conducted by Wright et al. [19]. Instead of determining the application protocols used by analyzing packet data and connection options like port numbers, the researchers of this study wanted to find a general method that classifies protocols based on features being available after different encryptions types and without factors that could be manually changed like port numbers. It started with creating a dataset containing the following network protocols/applications: SMTP, HTTP(S), FTP, SSH, Telnet, SMTP, and AOL instant messaging (AIM). The dataset was labeled based on port numbers available, and the encryption was simulated by applying a 64-byte padding to simulate the then-current encryption strength standard. Two different machine learning techniques were used to classify the network protocols, namely k-Nearest-Neighbours and Hidden-Markov-Model. With only post-encryption information being selected to be used for classifying, tuples were made containing direction, timing, and size for each packet. For 7-NN scores of 100% were reached for HTTP(S), SMTP-in, FTP, and TELNET. AIM, SMTP-out, and SSH scored between 75 and 83%. For the HMM the scores were lower, although the minimum was at 79%. Lastly, they tried to detect the number of sessions within an encrypted stream by using the HMM. With a 19% differentiation from the true number of connections for HTTPS streams, the results were quite positive.

Zhou et al. [20] compared different machine learning techniques to find out which of them detect VPN usage better and how well the classifiers can determine which applications were used within a TOR session. Both VPN and TOR can be considered as techniques that are using encrypted streams to encrypt all traffic including metadata between 2 points in the network. By calculating flow statistics which all take a factor of time and direction into account, a total of 23 features are created. 10 of them are selected using the Gini algorithm and used for training the classifiers, to prevent fitting on data with less value. When comparing these results with a principal component analysis it is found that by using the top 10 components, 98.4% of the data variance was kept. The (non)VPN classification was used to compare different classifiers which showed that especially neural networks, as well as random forests, perform best for this task. (with F1 scores of 92.5% and 97.1% respectively). For the application analysis, only the neural network classifier was used. Especially VoIP stood out (F1: 99.4%), followed by file transfers, video, and chat (95.9%, 95.5%, 94.3%) where the worst performing is audio traffic with 82.8%.

Since VPNs are a commonly used technique in network communications nowadays, it becomes harder to deliver a good working QoS to manage high-throughput data connections. Guo et al. [21] tries to identify 6 different protocols/use cases within an SSL VPN connection by making use of deep learning techniques. Using the public ISCXVPN2016 dataset, the data is split into individual streams based on a 5 tuple split (source IP, source port, destination IP, destination port, and transport layer protocol). Based on the size, the individual sample flows images of 39x39 pixels are made which is used as an input for different deep learning algorithms. Per algorithm the scores varying slightly although in general accuracy scores around 92% are reached for the 6 categories. Overall the protocols could be ranked in the following decreasing order VoIP > Email > P2P > Streaming > File Transfer > Chat.

Nevertheless, it should be said that in comparison to real VPN situations every protocol was split on the forehand, (instead of having to deal with combined channels as in the real world where multiple protocols can be used in one VPN stream,) which influenced the size trace compared to normal situations.

3.1.3. VPN usage detection

Although ports or standardized features could reveal the usage of OpenVPN protocols, this possibility disappears when those default settings are changed and payloads are encrypted. Pang et al. [2], shows that by basic analysis of statistical characteristics, advanced machine learning models are unnecessary to successfully find OpenVPN connections. This could help to identify VPN connections in high-speed backbone networks where it is not possible to capture all network packets and immediate results are needed. Looking at the specifics of the OpenVPN protocol (the diagram is presented in 3.1), it is shown that the first 5 bytes of the header are reserved for the 8 different operation codes. Saying we capture the first 10 packets only of a stream, the variable definitions from figure 3.2 could be used to create the following formula: $\{n_1 = 0, n_2 = 1, n_3 = 1, n_4 > 3, n_5 > 2\}$ to determine if a stream is probable an OpenVPN connection with a 99.9% accuracy. Although this research is not directly related to our study, it is included to provide a complete overview of the type of fingerprinting done concerning anonymity tools. In countries where VPN connections are prohibited to use, this could help governments to suppress citizens as well.

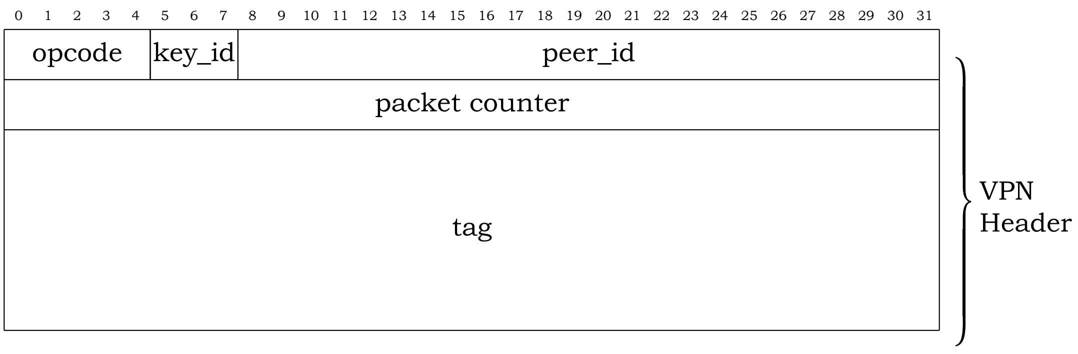
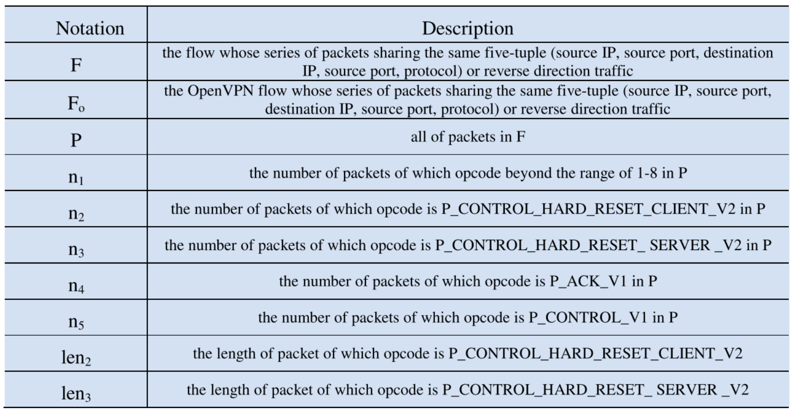


Figure 3.1: Openvpn packet header

3.1.4. Applications



Notation	Description
F	the flow whose series of packets sharing the same five-tuple (source IP, source port, destination IP, source port, protocol) or reverse direction traffic
F_o	the OpenVPN flow whose series of packets sharing the same five-tuple (source IP, source port, destination IP, source port, protocol) or reverse direction traffic
P	all of packets in F
n_1	the number of packets of which opcode beyond the range of 1-8 in P
n_2	the number of packets of which opcode is P_CONTROL_HARD_RESET_CLIENT_V2 in P
n_3	the number of packets of which opcode is P_CONTROL_HARD_RESET_ SERVER _V2 in P
n_4	the number of packets of which opcode is P_ACK_V1 in P
n_5	the number of packets of which opcode is P_CONTROL_V1 in P
len_2	the length of packet of which opcode is P_CONTROL_HARD_RESET_CLIENT_V2
len_3	the length of packet of which opcode is P_CONTROL_HARD_RESET_ SERVER _V2

Figure 3.2: Condition description for OpenVPN detection. [2]

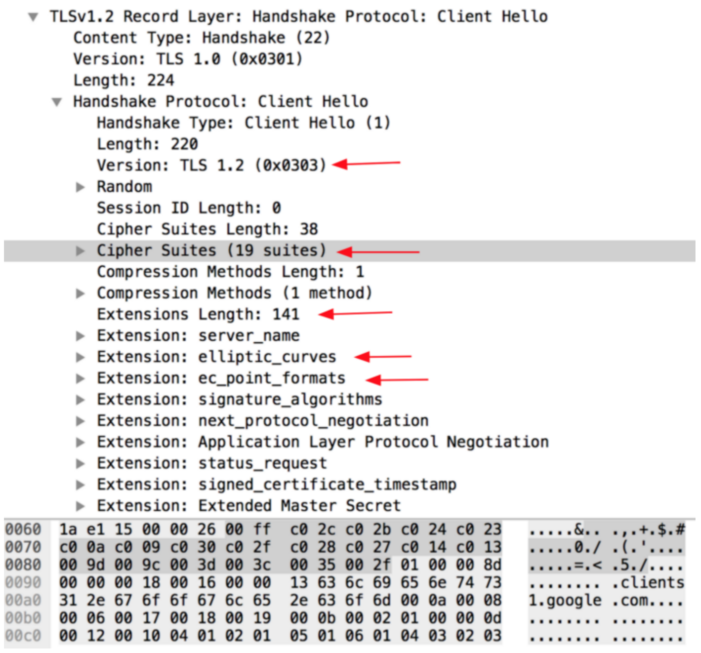
Where a lot of research is done looking at fingerprinting websites, modern browsers could influence the way this is done. Because web browsers are often relying on various APIs and different rendering engines, the fingerprints of a website could heavily change when visited using a different browser. Zhioua [22] is exploring the way different browsers influence the features used by earlier research projects which experimented with website fingerprinting. By redoing the previous work with different browsers the information variance per feature could be plotted. It comes clear that building features based on 1 browser could heavily vary when captured with another one for features like average run length, object, and packet interference ratios, object and packet order variances.

Not only browsers are handling the same traffic differently, but operating systems also tend to handle network connections differently as well. In 2017 Muehlstein et al. [23] tried to even further distinguish users by identifying websites, browsers, and operating systems all together based on the HTTPS streams. Although the usage of HTTPS raised during the past years, eavesdroppers still can easily leverage information about a targeted user. To prove fingerprinting is still an attack vector they picked 3 operating systems, 5 browsers, and 8 applications/websites to fingerprint. Based on earlier research, a feature set of 26 different stream statistics was used. During the research, another 27 features were created to be used alongside the basic set to further increase the accuracy. Using a support vector machine (SVM) an accuracy of 93.52% was reached using the basic set, where the extended set even increased this further towards a 96.06% accuracy score when predicting the combination of OS, browser, and application. It is hard to conclude whether or not operating systems are easier to classify than browsers or applications since the number of classes between those 3 is not equal.

On the android platform, Meijer [15] tries to classify android applications based on their streams as well. Although connections between apps and servers are encrypted, this doesn't mean that traffic patterns are gone, making it possible to track user behavior on mobile platforms. For this research, 500 different android apps are taken into account with data from 65 unique users. Using the main features; size, time, and direction, a total of 176 features could be created, mostly statistical values to describe the data bursts. During preliminary tests, random forests gave the highest scores on almost all metrics, followed by the convolutional neural network classifier. Although there were large differences in classifiability between apps, it was possible to reach an accuracy of 86% for the top 100 selected apps with real-world data. Furthermore, it was shown that the attacker needs to keep his classifiers up to date since the accuracy dropped by 7 percentage points over a time of 3 months.

3.1.5. Operating systems

Detecting the specific operating systems and their versions could help adversaries to set up specific attacks as well as forensic experts to easier identify all devices in the network. Where active scanning approaches could reveal the presence of an attacker or return false-negative results (e.g. when firewalls are blocking specific scan attempts), passive scanning could provide the listeners with a lot of information when knowing what to do with the observed data. An example of this is showed by Izadinia et al. [24] in 2006 when it was shown that it is possible to determine the operating system by inspecting parameters sent during the setup and usage of the tunnel. Differences in the protocol headers like the 'don't fragment' flag in the IP header, or the order of ISAKMP informational messages



```
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 224
  ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 220
      Version: TLS 1.2 (0x0303)
    ▶ Random
      Session ID Length: 0
      Cipher Suites Length: 38
    ▶ Cipher Suites (19 suites)
      Compression Methods Length: 1
    ▶ Compression Methods (1 method)
      Extensions Length: 141
    ▶ Extension: server_name
    ▶ Extension: elliptic_curves
    ▶ Extension: ec_point_formats
    ▶ Extension: signature_algorithms
    ▶ Extension: next_protocol_negotiation
    ▶ Extension: Application Layer Protocol Negotiation
    ▶ Extension: status_request
    ▶ Extension: signed_certificate_timestamp
    ▶ Extension: Extended Master Secret
0060  1a e1 15 00 00 26 00 ff  c0 2c c0 2b c0 24 c0 23   .....&.. .,.+.$.#
0070  c0 0a c0 09 c0 30 c0 2f  c0 28 c0 27 c0 14 c0 13   .....0./ .(.'....
0080  00 9d 00 9c 00 3d 00 3c  00 35 00 2f 01 00 00 8d   .....=.< .5./....
0090  00 00 00 18 00 16 00 00  13 63 6c 69 65 6e 74 73   ........ .clients
00a0  31 2e 67 6f 6f 67 6c 65  2e 63 6f 6d 00 0a 00 08   1.google .com....
00b0  00 06 00 17 00 18 00 19  00 0b 00 02 01 00 00 0d   ........ ........
00c0  00 12 00 10 04 01 02 01  05 01 06 01 04 03 02 03   ........ ........
```

Figure 3.3: TLS fields used for JA3-hash creation[3]

were manually found between different implementations. Although every platform was following the RFCs, the differences were found when the RFC left room interpretation like which side is first sending ISAKMP specific messages. In the end, 3 different operating systems could be easily distinguished by a set of 18 researched discriminants.

More recently Althouse et al. [3] came with a method to fingerprint operating systems, browsers, and/or applications by combining all cryptographic settings into a single hash. Since all kinds of traffic are encrypted with TLS often nowadays, it is harder to detect malicious content as well. By design, TLS is supporting various options to configure the secure channel for all platforms. (For example, SSL Version, Accepted Ciphers, List of Extensions, Elliptic Curves, and Elliptic Curve Formats). However, specific combinations can be unique for some programs making it interesting to use for fingerprinting. When using this to design a more generic fingerprint approach, it doesn't matter when malware switches to an unknown IP address or even using Twitter for C2 traffic. By concatenating all options and hashing that value by the researches so-called JA3 hash is created. This hash could be used to label all different kinds of traffic and could be used by network intrusion detection systems to detect Cobalt Strike for example.

3.2. Obfuscation methods

Not all research is focussing on how to fingerprint different things from data, finding obfuscation and preventive measures is explored as well. For example, Cai et al. [25] compares different defense techniques to prevent website fingerprinting in encrypted streams since they are a serious threat to privacy mechanisms. Although there already exist some defensive methods, most of them have a high overhead, poor security, or both. By exploring the current attacks and proposed solutions, it is possible to create an overview of the state of the art. According to this research, website fingerprinting is based on 4 major packet sequence features, namely unique packet lengths, packet length frequency, packet ordering, and interpacket timing. For each defense method, 2 classes are compared, 1 with the feature and defense method and 1 without the feature. When there is no discernible difference between the 2 classes, the defense is successful in hiding that feature. Defenses that are looked into are maximum padding, exponential padding, traffic morphing, HTTP obfuscation, background noise, and Buffered Fixed Length Obfuscator. It becomes clear that for maximum obfuscation effects, modifications of size and time are needed to hide as much as possible for the 4 picked features. Applying this directly to our research is hard since half of the methods are specifically designed for website fingerprinting instead of protocols.

In the earlier mentioned work from Meijer [15], obfuscation methods are compared as well. For this research, the fingerprinting was done with classifiers focusing on network streams as well. Although a really large feature set was available, the here-mentioned defenses are all focused on changing the size features. Proposed defenses are linear-, exponential-, MTU-, random- and mice elephant padding. The best working padding, which is lowering the information usable for classification, is padding to the MTU. Since this forces all packets to have the same size, the size feature is basically eliminated.

3.3. Conclusion

Within this chapter different studies have been discussed. Where the oldest ones were able to fingerprint different things quite easily, the rise of different encryption techniques forced researchers to use more advanced models as well. Nowadays, a lot of experiments are making use of statistical features of network streams as a common factor. Moreover, recent reports are using Neural Networks and Random Forest as the state of the art classification models since they often achieved the highest scores on different metrics when doing network traffic fingerprinting. Since some of the goals of this research include finding out which features are helping the most to make network protocol fingerprinting possible, the so-called black-box models are not suitable. This means that Neural Networks cannot be used in our research. Since Random Forests are ensembled white-box models, they are suitable after a little bit of tweaking is done to obtain feature importances.

Although protocol fingerprinting has been performed in a recent study, none of the aforementioned works explore the possibility of fingerprinting network protocols in combined encrypted streams. This is the exact thing that is observed when using a VPN in most real-life situations, making it impossible to split the traffic of different applications into different streams. However, all works involving VPNs are using a predefined dataset which was split per application on forehand. One of the goals of this research is to find out how much combined channels affect the classifiability of network protocols. Exploring this gap requires us to define new features as most previous techniques are not feasible to be used for the combined encrypted streams. However the base features size, direction, and interpacket timing are still available to create new features based on direct neighbors instead of streams.

Lastly, during this chapter different obfuscations are mentioned as well. These are a good basis set to apply on our dataset as well, although not all of the mentioned techniques are suitable since some methods are specifically designed for a protocol or application. Besides, in most experiments only size obfuscations are taken into account. This only covers half of the possible feature set, while timing obfuscations were not discussed as an option. Therefore those have to be designed later on to help not only answering the obfuscation question but also extending the basis for the feature importance part as well.

4 Methodology

To answer the research questions from chapter 1, different experiments need to be designed. However, a lot of steps should be taken first, before everything is in place to conclude on the research questions. Data is key to all actions taken along the entire process chain, although the exact form can change at each step. If we zoom in more on the process pipeline, we see that it consists of the following stages: generation (4.1), processing and enriching (4.2), modelling (4.3), classification (4.4), and evaluation (4.5). The creation and testing of obfuscation methods cover multiple steps and are therefore discussed in their section (4.6).

Requirements of a single step could influence both previous as well as next steps. An example of this is the requirement to collect both encrypted and unencrypted traffic to match the network protocols with the correct packets within VPN capture since we cannot decrypt those sessions. To do this systematically throughout this chapter, any solution presented in a single step that is needed for future steps contains a referring tag and be explained in the step where the requirement originates from.

By separating the processes, a lot of flexibility is gained. Each phase now stands apart from the rest, making it possible to modify separate parts without directly affect the functioning of others. For instance, capturing new network protocols in the data generation phase should not change the whole methodology of data classification part other than an additional network label.

4.1. Data generation

During the literature study, we found that most of the related research that involved VPNs were relying on datasets that contained streams that were already split per session or protocol. Part of this research is to find out the effects of dealing with the combined streams that VPNs create when transmitting data. Therefore we cannot use existing datasets but have to generate the data ourselves.

The data will be used by machine learning algorithms, and to train and verify them, they require the data to have the right labels. A labeled dataset means that for every data input the output should be known as well. As stated before, this means for this study that the extracted features per network packet are used as inputs and the output is the corresponding network protocol. Since it is hard to decrypt the VPN traffic without modifying VPN solutions, we need to make sure that we capture the unencrypted traffic right before it gets encrypted together with the encrypted VPN data. Below the more technical details of the data generated will be explained.

4.1.1. Technical setup

An important requirement of the data generation step is reproducibility. This can be ensured by generating the data within a virtualized environment like VMs (virtual machines) or containers, making it possible to always get the same environment and resources in exactly the same state as before, when launching a new VM or container. For the coding language used (Python), more software libraries regarding containers are available to use for automation tasks. That is why it was decided to use containers over VMs (using the Docker framework) as the basis for the data generation.

Capturing unencrypted and encrypted data is the next obstacle. Most of the time, VPN solutions create an additional network interface that simply processes all the traffic that is sent to the network interface of the VPN, the so-called route-based VPN setups. The additional network interface makes it possible to capture the unencrypted traffic right before it gets encrypted by the VPN interface itself. This has some advantages over capturing data from different interfaces with some intermediate hops in between, as in that situation more packets are shuffled than in the case of 2 consecutive network adapters. Although figure 4.1 will be fully explained in a few paragraphs, it is insightful to use that setup as an example for the just explained problem. For that scheme, it can be said that packet reordering happens more often when data capturing takes place on the first yellow and last purple adapter, than when capturing on both purple interfaces.

A policy-based VPN setup (like IPSec) however, is checking all traffic against a (policy) list without creating an additional network interface. This means that the unencrypted packet captures should be made some hops earlier, leading to more shuffled captures. Luckily the IPSec setup could be extended with a so-called Virtual Tunnel Interface (VTI). This adds an additional network interface, which makes the functioning of the IPSec tunnel comparable to route-based VPNs.

Unfortunately, this setup won't work within a container, since the incoming VPN packets towards the VTI won't be routed correctly through the docker network adapter. To overcome this issue, the VTI should be placed after the docker network adapter on the host. Unfortunately, this has the side effect that the data capturing process is now limited to 1 container per time. Running multiple containers at the same time would create uncontrollable mixed traces since the VPN connection is not set up inside the container. By adding another layer of virtualization this could be solved, since we could duplicate the host machines multiple times as well, making it possible to speed up the data generation process. By creating virtual machines in a large server center like vSphere, it is possible to create multiple host machines each with the same computing power, all being able to run containers inside them as well. One thing to note is that only capturing IPSec data on the host machine while others are captured inside the container could give an unfair advantage since the timing could be different. Therefore, all captures will be made on the host VM whereas the data will be generated inside a container.

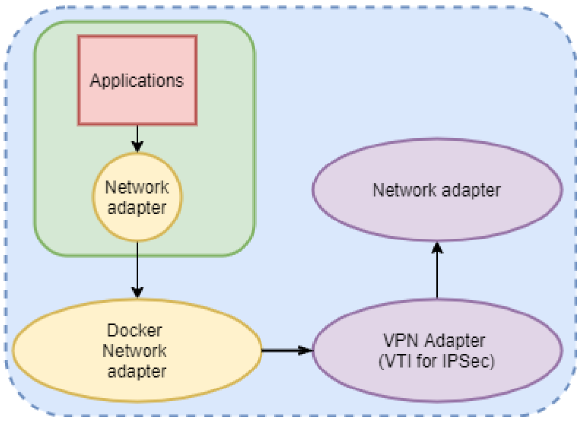


Figure 4.1: Data generation setup

A clear overview of the just discussed technical properties can be found in figure 4.1. The colors mean the following: the blue box is the host machine, purple ovals are the network cards that are used for data capturing and the green box is a container, running a specific application that will generate the wanted network traffic.

Segmentation offloading

Modern network interfaces try to handle packet segmentation themselves instead of the CPU. This can be useful to save resources and prevent latency caused by a busy CPU. Instead of receiving chunks as large as the MTU, they receive the original packets and split and resemble them by themselves. Those techniques are known as TCP segmentation offload (TSO) when applied to TCP and generic segmentation offload (GSO) for transmitting packets or Generic receive offload (GRO) for receiving packets. Making network captures with these modes turned on, leads to observing the reassembled packets instead of the segmented packets that were sent over the network. This is not useful when it is needed to match the packets with the encrypted ones. As disabling those modes will not influence the traffic content (since our hosts aren't doing other heavy calculations) and helps for better packet matching, these 3 modes are turned off during the data capture phase.

4.1.2. Protocol selection

The network protocol pool used for classification was chosen based on the daily use cases and the ability to create automated processes generating randomized data of the requested protocols. Since we want to train machine learning classifiers to be able to pinpoint the correct protocol per network packet on a small number of features, a lot of data is needed. During the data generation, we want to add as much randomization as possible to all tasks responsible for generating the data to make sure that the classifiers are training on protocol properties and not on repeating patterns of the data generation scripts. Based on those 2 requirements the following protocols were chosen: DNS, FTP, HTTP, ICMP, IMAP, POP, SIP/RTP, SMTP, and SSH. These protocols are mainly used for surfing the web, email traffic, simple administrative tools, and digital (video)calls and enough options are available to add to the generation scripts to enforce randomness in the tasks executed.

4.1.3. VPN protocol selection

The VPN protocols that are used during this research are chosen based on different reasons. The selected consumer VPNs are all commonly offered by VPN providers for private usage. Although the same goes for the enterprise protocols, the products are highly configurable and with their own, custom software for clients. As these products are very expensive, this pool was limited to the VPN enterprise products available to test with at the time of research.

In total, we compare 6 different VPN protocols, divided over a total of 8 VPN products (twice IPSec and SSL VPNs). They can be grouped based on different characteristics. In table 4.1 below, the different columns list the different characteristics. Most important are the differences between padded and nonpadded protocols, and the comparison between consumer and enterprise VPN protocols.

VPN-Product	**TCP/UDP**	**Padding**	**Consumer/Enterprise**	**Location**	**Notes**
OpenVPN UDP (NordVPN server)	UDP	N	Consumer	Cloud	
OpenVPN TCP (NordVPN server)	TCP	N	Consumer	Cloud	Packet combination due to Nagle's algorithm
Wireguard	UDP	Y (16B)	Consumer	Cloud	
IPSec (NordVPN server)	TCP	Y (16B)	Consumer	Cloud	Padding shifted (62, 78, 94, etc.)
IPsec (Pulse secure)	TCP	Y (16B)	Enterprise	On premise	Padding shifted (62, 78, 94, etc.)
DTLS (Cisco Anyconnect)	UDP	Y (16B)	Enterprise	On premise	Padding shifted (58, 74, 90, etc.)
SSL (Cisco Anyconnect)	TCP	N	Enterprise	On premise	
SSL (Fortigate)	TCP	N	Enterprise	On premise	

Table 4.1: Overview of the used VPN products/protocols and their specialties

4.1.4. Generating mixed traffic

In the real world, our computers are using multiple network protocols at the same time to do different tasks. For example, when starting your mail client, different protocols could be used like IMAP for mail syncing, SMTP for mail sending, and maybe POP as well when having multiple email accounts configured with different setups. To simulate the situations where multiple protocols are used, we have to generate mixed traffic captures first to answer parts of our research questions. In terms of the underlying setup, it means that multiple applications inside a container should start at the same time to create traffic streams. We only have to define a process to create them well-structured and in a reproducible manner.

The first challenge is to define a mixing rate that covers all scenarios, as the combination of programs running at the same time and their data transmissions patterns could vary a lot over time. To get comparable datasets for all VPN protocols, it was chosen to define 2 different mixing rates that both try to stabilize around a different percentage of mixed protocols at the same time. With 2 different mixing rates besides the non-mixed traffic dataset, namely the partially mixed traffic dataset with a mixing rate of 35% and heavily mixed traffic with a 70% mixing rate, we cover a range of realistic mixing rates that could occur in real life. The mixing rate itself is defined as the percentage of timeframes where 2 or more different network protocols occurred. For example, having a data capture of 40 seconds with a timeframe defined to be 1 second, we check for all 40 timeframes if the 2 or more different network protocols occurred in each timeframe. If this is the case in 28 of the 40 timeframes, the mixing rate would be 70%.

In practical terms, mixed traffic is created by starting an application then calculating a random delay with a maximum of x seconds (based on partially or heavily mixed traffic), and starting the next application. Since every application runs for a maximum of 5 seconds, all next applications that were started within that time can cause overlapped network traffic. Per capture 100 application tasks are started (which is large enough to have a nice distribution of the delays and small enough to debug faults or remove telemetry correctly). To achieve mixing rates of 35% and 70%, it was found that for most VPN protocols the random delays should be applied with a maximum of 10 and 2 seconds respectively.

4.2. Data processing and enriching

Right after the data generation phase comes the processing part. Which is done in 2 main steps. First of all, the raw data from the network captures need to be parsed, decapsulated, and matched with the right network protocol label accordingly. Thereafter new features can be created based on the basic interpreted data.

4.2.1. Encapsulation removal

A VPN always adds its own encapsulation around the normal network traffic which is an additional header and tail but comparable with encapsulations shown in figure 2.2. Which network layers are encapsulated exactly (internet or data link layer respectively), depends on whether it is a tunnel or a network tap VPN construction. For this research, tunnel VPN setups are used since those are used by consumer VPNs as well as remote access enterprise VPNs. A tunnel VPN setup implies that from the Internet-layer to the Application-layer could be present as the payload of the VPN traffic. Different VPN protocols, as well as cryptography options, determine the size of the encapsulation. Since the handshake of the connections is not captured, the encapsulation size should be known to set this variable in the data parser to correctly calculate the original payload size. It could be determined by manually matching the first packets of both encrypted and plaintext streams. This can even be automated by comparing different known padding sizes and calculating the lowest error rate.

4.2.2. Feature extraction

Since VPN clients route all traffic through a tunnel to the VPN server, IP and port numbers won't say anything about the inner traffic. Besides the content of the packets is meaningless as long as it is encrypted using a strong encryption algorithm. This leaves us with a total of three different features that can be extracted from the encrypted packets and have some meaning concerning the encrypted content, namely size, timing, and direction. Although not useful for the classification itself, packet numbers are extracted as well to help to solve the labeling problem in the next paragraph. Since it will be used for matching, the packet numbers from the plaintext captures (as well as the sizes and directions) need to be extracted as well. Furthermore, protocols from the plaintext files are needed for the classification tasks later on.

4.2.3. Labeling

Packet labeling happens based on the assumption of a one-to-one transmission (without reshuffling) of packets between the VPN and normal network interface. At the end of the processing phase, the size differences (between the encrypted and normal payloads) are calculated for every packet. When two consecutive packets have a wrong delta value, labels and plaintext sizes are swapped and the differences are calculated one more time. Only captures, where all delta sizes are correct during the first or second test, will be used for classification. This (almost certainly) ensures us that no packets are mislabelled and prevents the classifier from training on faulty labeled packets. It could only go wrong when multiple consecutive packets have the same size but are from different protocols and these are being mixed between the 2 network interfaces. Lastly, after all packets are matched, any unwanted information (like packets numbers) is removed so it cannot influence the classification process.

4.2.4. OpenVPN-TCP handling

OpenVPN-TCP and both SSL-VPNs are the only tested VPN protocols that run over TCP. In comparison to UDP, the TCP stack introduces a special algorithm to reduce the overhead of the IP/TCP layers by combining smaller packets when the Maximum-Transmission-Unit(MTU) is not reached yet. This is part of the so-called Nagle's algorithm. Due to this algorithm, there could a difference between the plaintext traffic and the final VPN traffic since packets were combined. Nevertheless, only OpenVPN-TCP shows signs that this algorithm was used. For this study, it was chosen to not disable this algorithm (since this introduces a difference compared to catching normal traffic on a line) but to split the payload afterward (where possible) using the payload length feature. As the packets were split afterward, the timestamp of these packets will be the same.

4.2.5. Feature aggregation

Although machine learning algorithms are really powerful when it comes to finding patterns in data, they won't be able to interpret the underlying meaning of the features or how they correlate in the real world. The earlier extracted features (size, time, and direction), could have a lot of additional information hidden about the original network protocol. As most network protocols operate according to their schemes, (for example answering in a specific format based on the request), consecutive packet sizes or inter-arrival times could all be based on the underlying network protocol. These patterns can be described in aggregated features like delta size or time. The features are created based on a maximum of 2 packets forwards and backward, as some network protocols (like DNS) don't have such long talking schemes, and because the chances are higher that both packets don't belong to the same data stream because of the application mixing.

Basically, all combinations possible with these features are created (shortly said: size, 4x size delta, 4x size delta directional, size opposite direction, 4x time delta, 4x time delta directional, direction, direction change). The only exception for the maximum 2 forwards and backward is the burst counter and burst counter maximum. Those can be much longer in theory, and they continue until a network packet is observed from the opposite direction. However, as long uni-directional streams are not common in highly mixed network environments but rather for single network protocols, these features automatically correct for mixed traffic. The aggregated features are calculated per data capture to overcome wrong values when calculating them after merging all the captures. A more detailed overview of all features (including the feature name, data type, and value range) can be found in appendix C.

4.3. Data modelling

Although machine learning models and their usage are strongly combined, this section will focus on the classification and data models used for this study whereas the next section (Data classification) will focus on how to use them in combination with our data.

Talking about machine learning, the classifiers could be grouped in various ways. One important division for this research is based on the ability to determine on which features and content an algorithm decide, and is known as white, black, or hybrid box models. White box models are fully transparent, meaning that one could fully trace back on which features the model made a decision. Black box models often contain multiple layers with different parameters that are interconnected. Those models are trained through feedback learning and hard to understand how they evolved. Hybrid models are some kind of a combination of both worlds. Where the performance of black-box models is combined with the explainability of white-box models. During the literature study, it was found that most studies are using random forest, neural networks, or Markov models as classifiers for their data. However, neural networks won't allow us to trace back the workings or extract the feature importance. It was therefore chosen to continue with both random forest and Markov models.

4.3.1. Random Forest

A random forest is an ensemble classifier build by combining multiple decision trees. Those decision trees are each build with a random subset from the data features available. Each tree is independently trained with the selected features. When a data point should be classified, a majority vote could be used to determine the final class. A random forest is considered to be a hybrid box model since the permutation of data features for every tree is done randomly and this would not always give clear insights. However, when having a large number of trees and combining both feature importances and splitting values, some conclusions could be drawn.

4.3.2. Markov model

Markov chains are stochastic models named after a Russian mathematician Andrei A. Markov. It describes the possibilities of events occurring, where the probability is only based on previous events. Concretely said, Markov models can be described as state diagram models where state transitions have a certain likelihood of occurring. The sum of all outgoing transitions should be 1. As the name state diagram implies, the data should be a time series. By modeling a specific feature for every class, the likelihood of belonging to one class could be calculated when taking a certain amount of state transitions and take the highest likelihood. The hardest part is abstracting the data in such a way that the states and their transitions cover the most important information. This is done by creating buckets based on the entropy of the data as described below. The Markov model itself could be used as a stand-alone classifier or its outcome could be added as a feature to the data for other classifiers.

Entropy based buckets

The only variables for encapsulated server-client communication, independent of the location where the traffic was captured, are size and direction. Time, however, is dependent on internet speed and processing power for packet reassembling, the location used for capturing the data traffic, and more. Based on the size and direction information, it would be possible to create states for every protocol. Not wanting to have a state for every size possible, a solution is needed to abstract the data. Since packet sizes are not homogeneously distributed, it won't be useful to create equal splits over the size domain. By analyzing several captures of a VPN protocol with randomly generated traffic, it is possible to get a good representation of the size distribution per VPN protocol. The basic procedure is quite simple. For example for the size feature:

When n buckets are wanted and x packets are available, first create buckets for every size which has more than x/n packets in the set and remove those packets from the set. Then all packet sizes left are ordered in a list. When one wants to create the buckets left (let's say m), the first split will be placed after $100/m$ percent. Then all numbers lower or equal to the value of the first split will be removed, and this sequence will be repeated for the next buckets by taking $100/(m-1)$, etc. This way the buckets have proper ranges to spread the packets as equally as possible.

The results from these mappings can be found in appendix B.3.2 and B.3.

Markov based features

With the created buckets for packet sizes and (inter-arrival) timings, it is possible to create Markov models. States are formed from both the buckets and direction and formed according to the following syntax: $direction\_bucket$. From those states, 3 different types of features could be created, all with 2 'flavors'. Those state transitions could be calculated based on the chances of the local state (sums up to 1), or with the chances calculated over the whole model. The first option eliminates the bias of being in a rare state since it is only looking at the local state transitions. The second flavor will include this, which could help to compare two different models where both states do occur.

The 3 different types of features are 3-Markov, 5-Markov, and maximum-likelihood-Markov scores. The 3-Markov will be the multiplication of the previous and next state transition chance, where the 5-Markov looks 2 states back and 2 forwards, meaning that its score is based on 4 transitions instead of 2. The maximum-likelihood score is the highest single transition score of the 4 transitions used in the 5-Markov feature. Those 3 features are calculated for every protocol and both flavors, giving a total of $9 * 3 * 2 = 54$ new features.

4.4. Data classification

With the machine learning models defined, we have to decide how to use them exactly for our study. Since this involves multiple and different classes, different approaches could be taken. Furthermore, cross-validation needs to be in place to make sure the data is not overfitted on parts of the dataset.

4.4.1. Multiclass classification vs One-hot encoding classification

When trying to solve a classification problem there are multiple approaches. Multiclass classification means 1 classifier to distinguish all labels. Since this setup has to consider all labels, it gives a better overview of which features are more important in general. This is useful when considering protocol fingerprinting in general and why it works. One-hot classification is done by training n classifiers, 1 per protocol, each predicting if something belongs to a class or not. Besides having 9 classifiers for our study (1 per-protocol vs 1 in total), so a slightly more accurate classification, one could imagine this setup would cost more time to train. On the other hand, this setup can be used to better understand which features of a network protocol could help to distinguish it from others.

4.4.2. Balanced datasets

To make sure that every network protocol is evenly represented, we had to generate more captures than needed. This is because the mixed traffic captures are randomly generated and some network protocols have shorter conversations by default. After all the features are created, the corresponding data captures are merged into 1 dataset. From this dataset, 10k packets of each network protocol are randomly picked and stored in a new dataset. Thereafter the new dataset is shuffled to randomly distribute the network packets over the dataset. A positive side effect of these shuffles is that the chance that several consecutive packets are next to each other is very small, preventing the model from local overfitting on certain data stream patterns.

4.4.3. K-fold cross-validation

One way to make sure the classifier is not overfitted on specific data is to train and test it on multiple data sources. When only having a single data source, one could apply k-fold cross-validation to achieve the same objective. The data is divided into a training and test part just as normal, but now the train and test phase is repeated k times. Every time the data will be shifted $1/k$. An example of this is shown in figure 4.2. In this figure, a 5-fold validation is applied, with a training size of 80%. The blue part represents the test part, and as can be seen, this is shifted to a new part of the data every time.

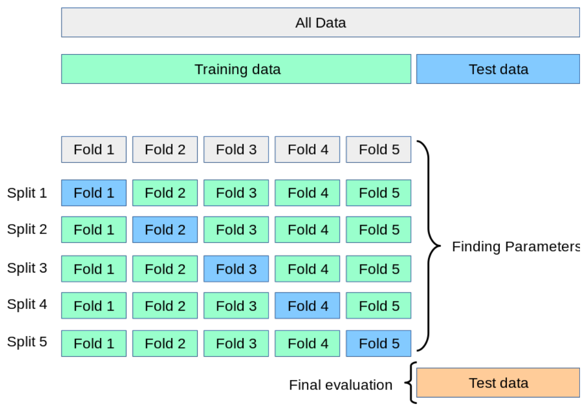


Figure 4.2: k-fold cross validation [4]

4.4.4. Markov

As mentioned earlier, the Markov chains could be used as a classifier or as features for other classifiers. When used as a solo classifier, the decision is based on the highest Markov score of a single type feature ((3-chain, 5-chain or Maximum-likelihood type) and (local or total chances)) for every protocol, meaning that one takes the highest score out of 9. When the probabilities are used as features for the next classifier, all scores could be used, and the decision is made by the next classifier instead of the Markov model.

4.5. Evaluation

There are many ways to compare the workings of a classifier, often based on the needs. A related, real-world example would be the covid-19 test, where the true positives and false negatives numbers are way more important than false positives. However, for our study where we want to compare all aspects of the classifier, not one metric could cover all. A first general useful metric would be the accuracy score. As the name suggest this indicates the overall accuracy and is calculated by $accuracy = \frac{TP + TN}{TP + TN + FP + FN}$. As we work with a balanced dataset, and in general we are most interested in the overall performance of the classifiers, the accuracy is the most important metric for this study.

When looking at true/false positives and negatives more precisely, the positive numbers are more useful than the negative ones, since our study has to deal with multiple categories and not a binary decision. Indicators that are focused on the positive rates are precision, recall, and the f1-score which combines both numbers. Where the precision number tries to determine the amount of correctly identified items amongst all positively classified items, recall tries to determine the number of correctly classified positives against all real positives. In terms of formula, this means: $precision = \frac{TP}{TP + FP}$ and $recall = \frac{TP}{TP + FN}$. To get a complete overview of how the classifier performs overall, these metrics should be calculated for every network protocol individually as well.

4.5.1. Feature Importance

Another output that is used to answer some of the research questions is feature importance. Although using an ensemble classifier for the main classification task, there are ways to overcome the fact that ensemble classifiers (like random forests) are using permutations of the feature set and they, therefore, might not use all features across the whole model evenly. Since random forests make use of decision trees, and our classifiers consist of a large number of trees, it is still possible to retrieve representative feature importance. This can be achieved by calculating the percentage of splits in a tree linked to a specific feature and averaging this for the number of decision trees used. This way we get an (averaged) score of how useful a feature is for classifying the network protocols correctly.

4.6. Obfuscation methods

Not only it is interesting to see if underlying patterns of network protocol make it possible to fingerprint them while in an encrypted stream, finding methods to prevent this from being possible is even so. Nevertheless, not many solutions are being developed, which makes it hard to compare them side by side in practical experiments. Without the full implementations, obfuscated data cannot be generated in the data generation phase, and has to be simulated by applying the obfuscation effects on the data after it was processed.

Exploring possible mitigations might be done by first determining what kind of features it should try to affect. Looking at encrypted traffic there is not so much information left, besides the encrypted data. Only the size, timing, and direction are easy features to obtain and useful for classification. Having the restriction of simulating the obfuscation afterward, lowering the Maximum Transmission Unit would not be an option for example. As this mitigation simply means that the network should send more packets to deliver the same amount of bytes, it is practically impossible to do this manually, as it would require us to exactly simulate how the network segments packets and applying correct delays as well. What is left are the paddings and delays for time and size features.

Size is often the main feature to focus on since this is independent of the platform, application, or external factors like network speed. The methods summarized in the previous chapter are straightforward to apply afterward where it often relies on the distribution or special sizes. To influence the timing feature new methods should be created, although the basic principles could be picked from the size obfuscations. This leads to minimal timeslots for sending packets based on the timing distributions or adding random delays. Since some studies suggest that network packet delays are based on Pareto distributions instead of a Gaussian one, both are compared as well.

4.6.1. Size related measures

Size-related measures will, as the name suggests, influence the size of the packets to limit the information that is obtained and used for classification. Since packet sizes are not equally distributed nor looking the same for different traffic directions, various setups are tested based on methods found in previous research. By applying these obfuscations to the data afterward, we could measure the influence on the classifiers for the size, where the timing is not influenced although this could have happened in the real world when there are places with a limited internet speed.

Strive for x% homogeneity by padding

When inspecting the entropy bucket plots, used for the Markov chains as well, the distribution tells a lot about common packet sizes. This method increases the lower packet size of packets towards certain points, that all packets below are shown as 1 class. Based on this plot, we took the cumulative packet distribution starting from the smallest packet size and by taking steps of 25% per time on this line, 4 datasets are created where at least x% of the dataset has the same packet size. A 100% homogeneity can be achieved by padding all packets to the MTU.

Mice elephant method

This method tries to increase the homogeneity of the packet sizes but also keep the overhead as low as possible. This is done by picking 2 packet sizes, of which one is the MTU where the other is quite smaller but large enough to cover most of the smaller packet sizes. When observing that more than 75% of the packets have a size lower or equal to 128 bytes, it prevents a lot of overhead using this method.

Exponential or Linear padding

Exponential padding is done by applying padding to the packet sizes towards the next power of 2. Effectively having 6 different packet sizes (64 -> MTU since almost every packet is larger than 32 bytes).

Linear padding instead, is done padding the packet sizes towards the next multiple of 128. Depending on the MTU there are 11 - 12 different packet sizes. In contrast to exponential padding, the smallest packet size is 128. Where (according to the distribution plot) more than 75% is below the 128 bytes, this could be to the advantage of linear padding above exponential padding.

4.6.2. Timing related measures

Influencing the timing could be done in different ways. Creating a tight timing scheme or a more chaotic situation both belong to the options. Although the last one is theoretically possible to test with the so-called Traffic-Controlpackage on Linux, it was chosen to manipulate the data after creation. This gave the opportunity to quickly experiment with different methods and is now equal for both methods since time scheme padding was not included for Linux. However, packet retransmissions due to long packet queueing are not taken into account nor simulated.

Minimum timing based on the timing distribution

Using the same method as the homogeneity size padding, entropy buckets are used to determine the distribution of the timing delta. Thereafter, different levels are chosen and applied. By creating a minimum timing, packets are held till this minimum is over. This way there are fewer differences in timing due to underlying protocol patterns.

Another variance could be created by applying the minimum timing per direction by tracking 2 timing variables. This way an answer can be sent immediately when the responder was already quiet for a long time. Since we don't want to influence the order by doing so, since the timing was artificially influenced afterward, this might not fully represent a real-world timing obfuscation. However, since a lot of packages are TCP acknowledgments, it is questionable which percentage of the packages would be different.

Additional random timing

Instead of striving for homogeneity, applying random delays could also lower fingerprintability. We have based the choice for the distributions that will generate random delays on the Linux traffic control package and literature. This showed that Pareto, as well as Gaussian distributions, are used most often. For every packet, a random delay is added before sending, where those 2 distributions are applied separately from each other.

5 Results

Having the setup in place, a lot of questions should be answered. By doing this we want to fill the gap of research regarding the effect of combined channels of VPNs on fingerprintability. It will start by exploring this effect in 3 different mixing ratios, after which we will diverge different VPN protocols. Then the different input features will be studied followed by the different network protocols used before we conclude on different possible obfuscation methods to lower the classifiability.

5.1. Mixed traffic comparison

One of the biggest gaps in comparison to current research is the fact that only the streams of VPN connections running a single application per time are studied. One of the most important features in terms of normal VPN behavior lies in the fact that a VPN uses a single encrypted tunnel to transfer data from different applications back and forth. This makes it impossible to retrieve the original (single stream) per application from a combined channel. So before going into more details on the differences between VPN protocols, network protocols, or feature importances, it is important to find out to what extent mixed traffic sessions would influence the ability to classify network protocols.

As explained in the methodology, we have created 3 types of datasets containing different mixing rates for network traffic to see how all the experiments would perform in an ideal situation or a more realistic scenario as observed in the real world. The distributions of the accuracy scores per type of mixed traffic are plotted using boxplots in figure 5.1. The colored area of a box shows the 25 to 75 percentile of the data including the median. The whiskers are set on the smallest and largest values that are within 1.5 times the interquartile range (p75-p25) calculated from the boundaries of the box. This method is can be used to determine which results are considered normal when they are within this range or an outlier when they are outside of it. On a perfect Gaussian distribution, only about 0.7% of the total distribution would be considered as an outlier.

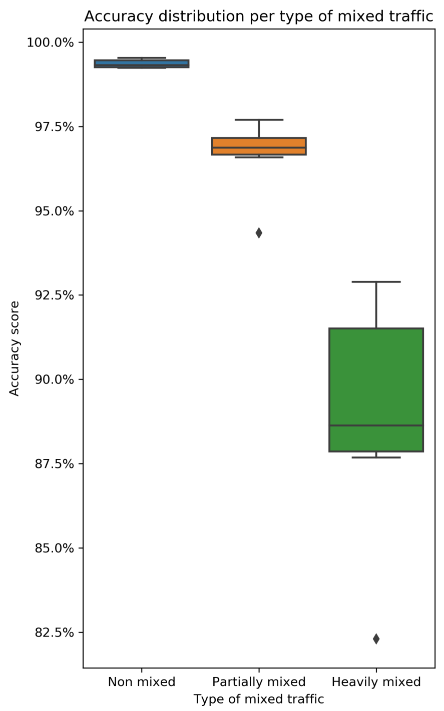


Figure 5.1: Accuracy score distribution per type of mixed traffic

Based on the plotted boxes, we can conclude several things about different types of mixed traffic. First of all, it seems that non-mixed traffic achieved a near-perfect accuracy score, which is in line with both Zhou et al. [20] and Wright et al. [19], who both achieved over 90% accuracy scores. A common factor for all 3 experiments is the usage of a filtered non-mixed dataset. This means that all datasets were filtered in such a way to contain non-mixed streams per network protocol. The achieved score practically implies that in a clean environment (implying almost no telemetry nor multiple applications running in parallel), the usage of a VPN won't be able to successfully hide the network protocols used.

Looking at the partially mixed traffic, the accuracy begins to drop a little bit, including a higher variance between the scores achieved. Still, an average accuracy of around 97% is achieved, meaning that per 100 network packets, 97 can be correctly identified to which network protocol they belong. The higher variance means that the scores of different VPN protocols are less close together. Based on the characteristics per VPN protocol, as listed in the methodology as well, different groups of VPN protocols can be formed. As we classify based on time and size patterns of network protocols, differences in how VPNs are handling packets size- and timing-wise could have a large impact on the easiness of classifying network protocols. A more detailed look into the accuracies per VPN protocol makes it possible to conclude which characteristics are responsible for the variance of accuracy scores, which will be done in the next section.

Taking the heavily mixed traffic, it has an even larger variance although with the same amount of outliers. This is interesting as having a dense centered distribution where only the variance towards the outer scores increased, would likely have resulted in a smaller interquartile range with more outliers. This means that the set of VPN protocols can be split into a small number of groups, (instead of being distributed equally over the landscape,) as the interquartile range (covering 75% of the distribution) grew without having an increased number of outliers. As we will focus on the VPN protocols individually in the next section, we will discuss their scores, observed groups, and outliers there.

As the variance of the heavily mixed traffic is the largest, this traffic distribution is the most useful to compare individual differences of network protocols, feature importances for classification, etc. Furthermore, as the heavily mixed traffic is probably the upper bound of the real world's mixing rate, it will represent the lower bound of the scores that can be achieved when having the same amount of network protocols. Therefore, heavily mixed traffic will be used by default in the next paragraphs for comparisons. One thing to notice as well are the outliers for both partially and heavily mixed traffic. This appears to be OpenVPN-TCP which will be researched later when trying to explain this outcome in section 5.4.

Although we achieve such a high score for heavily mixed traffic, the accuracy would likely drop in real life, as the classifier should be able to classify more network protocols correctly. However, this has to be researched in the future to conclude about those numbers. Nevertheless, it is likely that detecting if some network protocols are used in a stream is easier than classifying each packet correctly. As the lowest-scoring VPN protocol for heavy mixed traffic still achieves a score of 82% when classifying each packet, changing the target can increase the score and compensate when a larger number of network protocols need to be classified. This could have a huge impact in situations where governmental organizations forbid the usage of certain protocols and want to monitor the usage of them.

Talking about real-life scenarios, another factor should be discussed. As all captures were made in a clean environment, there is no unrelated background noise in any of them, as we want to focus on the underlying workings as well. However, it should be said that Windows seems to send large amounts of telemetry by default, making it harder to capture clear network traces. This is less of an influence for enterprise environments, as those machines often send less telemetry, as parts of them are blocked by managed group policies. Operating systems like Ubuntu, Debian, and CentOS are nearly clear of telemetry, which makes it easier to successfully fingerprint connections them. Ironically, the more privacy-concerned people, who are blocking telemetry or using open-source operating systems in combination with a VPN, are probably easier to fingerprint than the ones who are using a VPN on a default windows machine just to bypass Netflix' geographical restrictions.

In the next sections, we will try to figure out why the fingerprinting is working, which characteristics are of most influence, and how this scenario could be possibly prevented. As it is therefore needed to differentiate VPN protocols based on their settings, it is better to use datasets with higher mixing rates as they have the highest variance in accuracy scores.

5.2. VPN protocol comparison

As we just saw that heavily mixed traffic has the largest variance between VPN protocols, we take this as a starting point for comparing individual VPN protocols. Figure 5.2 shows the accuracy scores for every VPN protocol which we can use to determine the order of performance. As can be seen in the plot, the best performing VPN protocols are the SSL-based protocols scoring around the 93% when using 2000 packets per protocol for the training phase. They are followed by OpenVPN-UDP (91%) and DTLS (89%) thereafter. Then comes both IPSec variants and Wireguard (all around 88%), after which it ends by the OpenVPN-TCP protocol with 82% accuracy.

Remarkably, the differences between the VPN protocols seem to be caused by applying padding to every packet at first glance. However, both OpenVPN protocols score lower than comparable protocols and OpenVPN-TCP is even considered to be an outlier based on the boxplot of the previous section (5.1). The fact that non-padded protocols score higher than padded protocols will likely be influenced by the information embedded in the size features since that is the biggest difference between both groups. This hypothesis will be researched more extensively in section 5.3.

Another thing that comes to mind is the fact that there is no real difference between consumer and enterprise VPN solutions. Thinking of it at first, it might seem logical that advanced enterprise products are more security-focused and provide better protection against possible information leakages than open-source variants. On the other side, (as will be discussed in the countermeasures section as well,) every mitigation often comes with a time or bandwidth overhead, causing to introduce problems otherwise. Something you want to avoid at all means in enterprise environments.

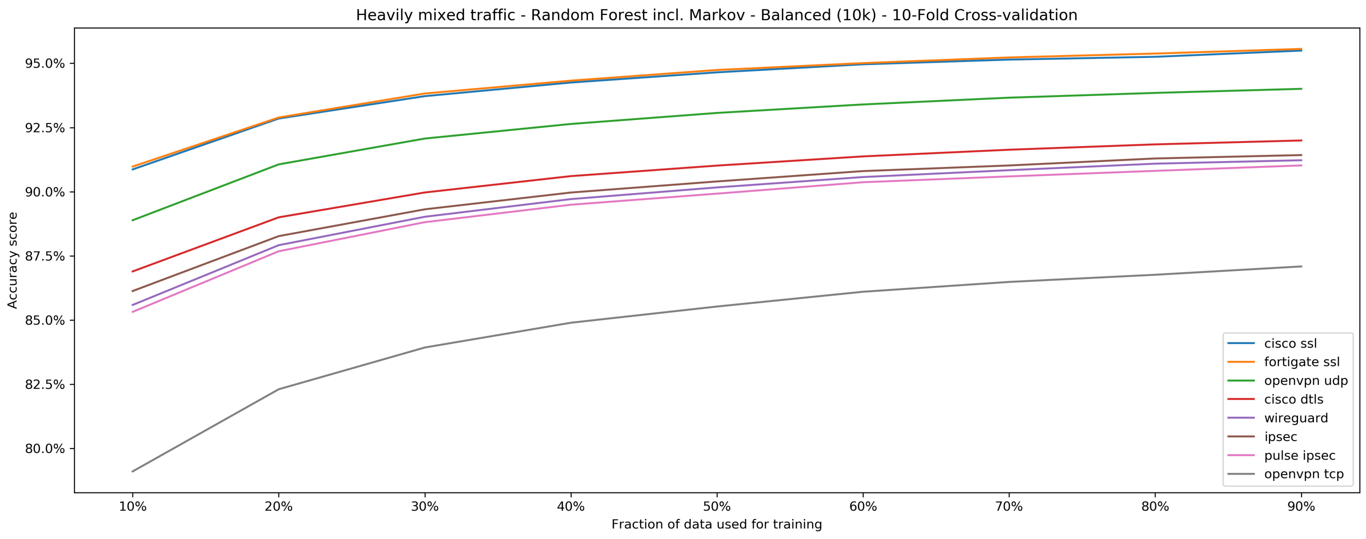


Figure 5.2: Accuracy scores for heavily mixed traffic

Taking a better look at the pool of padded VPN protocols, there does not seem to be a significant difference between the different protocols nor solutions. All the padded protocols are rounded up to the next 16 bytes, however, Cisco's DTLS is shifted in comparison to both IPSec captures. The latter does not seem to have an effect, although this should be verified with different VPN protocols as well. Moreover, different padding schemes could be studied in future work, although those used as an obfuscation (which are more extreme), will be discussed in section 5.7.1.

Another interesting observation is the fact that it is not possible to see a clear effect of on-premise versus cloud-hosted VPNs (overview in table 4.1). This is an interesting observation as normally packets towards a cloud-hosted VPN have to travel a longer route with the overhead attached. Although it was expected that this would influence the timing features and packet ordering drastically, this seems not to result in lower accuracy. This could be interesting as it could mean that the placement of a network sniffer along the route would not have a large impact on the fingerprintability of the network protocols, making it easier to perform this attack. However, we couldn't verify this hypothesis during this research due to the design of the setup.

Although the differences between on-premise and cloud-hosted VPNs do not seem to matter significantly, we will pick Cisco SSL and DTLS when wanting to compare padded and non-padded protocols more specifically to rule out any other influences. Since both implementations are hosted on the same firewall and use the same software to connect, the routes, resources, and influences other than protocol-specific differences will be ruled out. As the accuracy scores seem to be in line with the other solutions as well, it is possible to generalize findings based on padded versus non-padded protocols. This makes the graphs in the next sections easier to read and understand.

5.3. Feature importance

As we want to know why the fingerprinting of the network protocols works as well as it does, it is interesting to look at the importance of the different features used as an input for the classifier. Since the Markov models were only based on size buckets and not even all size features we focus on the feature importances from the random forest classifiers. As random forests are ensembled models from different decision trees which are all using a subset of the features, we will obtain a general feature importance by averaging over all the individual importances, which will level out the permutations of features as well.

Taking both single traffic as well as heavily mixed traffic, we have the two extreme variants for the feature importance. Where the single traffic feature importance plot will describe the optimal way of determining a network protocol, heavily mixed traffic will show us the more realistic scenario for real-world usage as most of the time the encrypted channels will be used by multiple network protocols within a timeframe. Both are of great value to understand why this attack works and how dependent this is on fluctuating features in busier situations.

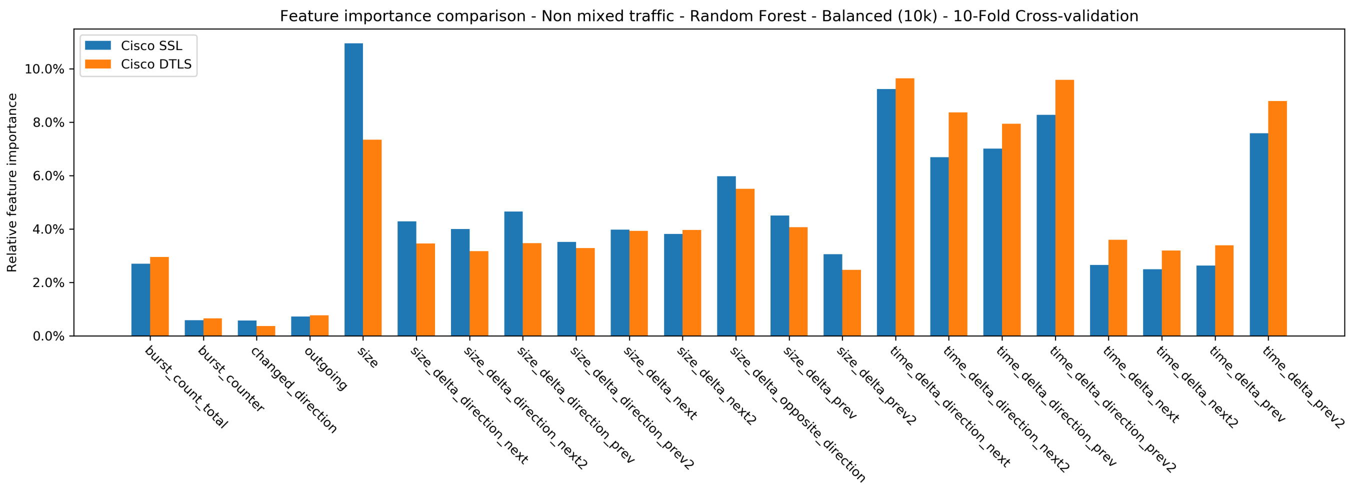


Figure 5.3: Comparison of the feature importance for single mixed traffic

In figure 5.3 the feature distribution for non-mixed traffic is shown for both SSL as well as DTLS traffic (representing non-padded and padded VPN protocols). The sum of all importances should be 1 as this equals 100%. For both protocols, there is a wide range of features used for classification, which varies mostly between the 4% and 10% importance. A few interesting things can be observed.

First of all, it seems that in general SSL is relying somewhat more on size features whereas the importance of timing features is higher for DTLS. This is caused by the fact that padded VPN protocols have less unique information embed in comparison to non-padded protocols. As the information ratio drops, the classifier has to find other useful features to make decisions on, which could explain the rise of timing features. This will e explored later as well.

Secondly, it is interesting to see that for the SSL VPN the size feature is followed by multiple timing features before other size-related features. This could be caused by the fact that every size feature is based on the packet size itself, information that is already known. Whereas the time deltas are completely new information as the timestamp is not a suitable feature itself. However, this is bounded by the fact that the timing features for single traffic are not interfered with by other traffic (noise).

Lastly, there is a large difference between directional and non-directional timing features, where the first one is more useful for classification. As the directional timing features are less influenced by traffic delays (especially the outgoing packets) and network protocols could hold packet answers to improve the channel performance, directional timing deltas are probably more trustworthy and useful for the classifiers.

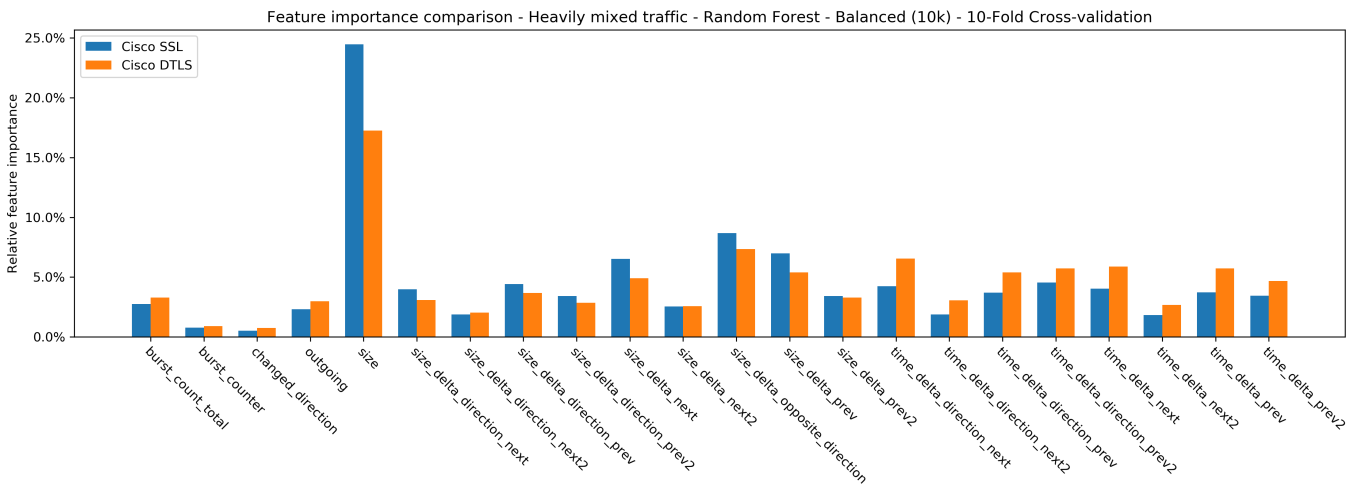


Figure 5.4: Comparison of the feature importance for heavily mixed traffic

Looking at the heavily mixed importances in 5.4, the whole plot seems to be shifted. The variance grew largely, resulting in a spike for the 'size' feature where the differences between other features seem to have shrunk. This can be logically explained by the fact that most features are built on the relationship with the direct neighbors (like delta time or size). As mixed traffic mixes up different network protocols, these features lose in value as 2 consecutive packets do not always belong to the same protocol anymore. Since the size is packet specific, this is a reliable feature to use for classification. In comparison to non-mixed traffic, the timing features dropped even harder, as multiple size-related features are more important than the timing ones. The shift to higher time importance for DTLS in comparison to SSL still seems to happen, although time is not the most important feature anymore for DTLS.

The cause for the differences between padded and non-padded VPN protocols might be found in the lower entropy for size since padded protocols do contain less unique information per packet due to their rounded sizes. To back this hypothesis, 2 new plots were made based on the accuracy score after classifying on only timing or size features only. The results can be found in figures 5.5 and 5.6.

In the first plot (5.5), which was created by calculating the accuracy scores when only using timing features, the accuracy scores for almost all VPN protocols are close together. However when using size features only, as shown in plot 5.6, padded and non-padded protocols are separated from each other. This indicates that for padded protocols, the size features are less useful indeed, which increases the dependency on time-related features

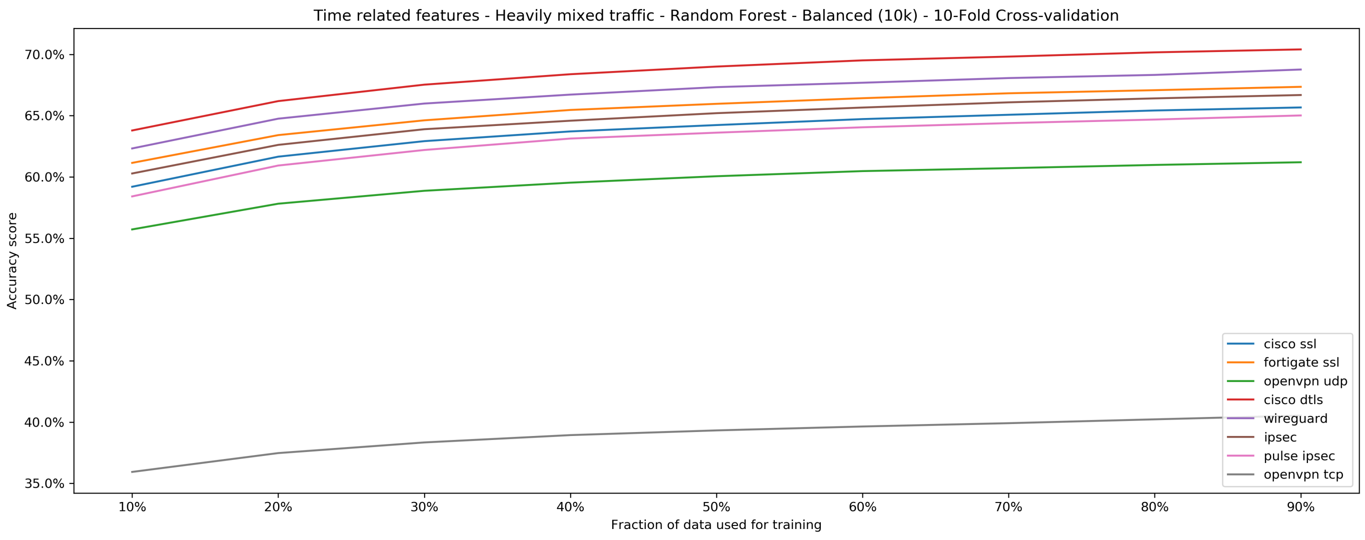


Figure 5.5: Accuracy comparison with only time related features for heavily mixed traffic

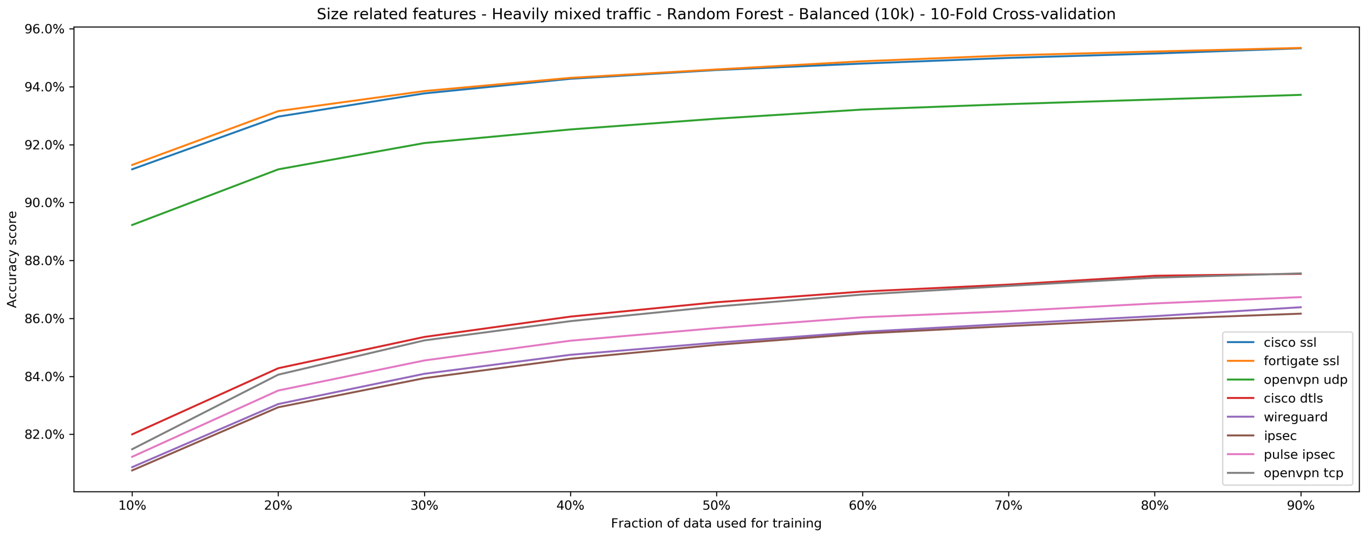


Figure 5.6: Accuracy comparison with only size related features for heavily mixed traffic

since the total sum should be 1. Moreover, since the size is the most important feature for mixed traffic as it doesn't rely on the packet ordering and mixing, this explains why the differences between padded and non-padded VPN grow as the mixing rate does the same.

5.4. OpenVPN

As was briefly mentioned in section 5.2 when discussing the boxplots and an outlier was shown, it seems that both OpenVPN protocols score lower than their direct peers. The question raises why both OpenVPN protocols are performing worse compared to the protocols that share the same basic properties. Since the previous section (5.3) showed us that both OpenVPN protocols perform worse than those with relatable characteristics and even when using time or size features only, the differences should be caused by something different. The first starting point might be the mixing rate, as the variance in accuracy scores is growing as the mixing rate increases between partially and heavily mixed traffic.

As explained in the methodology, our definition of the mixing rate is based on the following formula: for every delta t in range n determine if 2 or more network protocols are interfering with each other. If so, this time delta counts as a 1, otherwise a 0. Then take the average of all t in range n. Tables 5.1 and 5.2 show the mixing rates calculated for different timing intervals (t). As can be seen in the tables, the mixing rates are consequently higher for both OpenVPN protocols while all other VPN protocols are much closer together. Even though all datasets are generated in the same way, namely launching 100 tasks per capture, mixing rates could still differ when on average more tasks are still running when new ones are launched. In this situation it could be caused by the fact that OpenVPN processes the data slower, making tasks run for a longer period, causing more traffic to become mixed, which in the end leads to higher mixing rates.

VPN Protocol	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
cisco_ssl	0.246	0.361	0.402	0.425	0.452	0.460	0.473	0.492	0.502	0.515
fortigate_ssl	0.228	0.343	0.384	0.415	0.440	0.453	0.468	0.483	0.496	0.506
openvpn_udp	0.194	0.305	0.383	0.437	0.455	0.480	0.498	0.514	0.527	0.544
cisco_dtls	0.238	0.352	0.392	0.425	0.449	0.458	0.474	0.488	0.504	0.514
wireguard	0.269	0.380	0.416	0.443	0.466	0.477	0.490	0.505	0.515	0.526
ipsec	0.224	0.341	0.405	0.429	0.456	0.467	0.487	0.499	0.511	0.523
pulse_ipsec	0.276	0.384	0.418	0.442	0.462	0.475	0.489	0.499	0.513	0.525
openvpn_tcp	0.190	0.299	0.371	0.428	0.466	0.509	0.539	0.559	0.572	0.586

Table 5.1: Mixing rate of partially mixed traffic per protocol

VPN Protocol	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
cisco_ssl	0.568	0.664	0.718	0.748	0.774	0.777	0.788	0.800	0.796	0.804
fortigate_ssl	0.558	0.663	0.713	0.746	0.774	0.789	0.795	0.804	0.809	0.815
openvpn_udp	0.655	0.732	0.775	0.799	0.817	0.824	0.830	0.834	0.837	0.839
cisco_dtls	0.565	0.668	0.715	0.747	0.768	0.776	0.787	0.792	0.794	0.794
wireguard	0.594	0.697	0.741	0.770	0.790	0.798	0.805	0.810	0.813	0.818
ipsec	0.596	0.697	0.744	0.770	0.795	0.803	0.811	0.814	0.813	0.818
pulse_ipsec	0.590	0.692	0.736	0.761	0.780	0.790	0.792	0.797	0.799	0.799
openvpn_tcp	0.715	0.782	0.812	0.830	0.839	0.845	0.850	0.852	0.855	0.857

Table 5.2: Mixing rate of heavily mixed traffic per protocol

To back this hypothesis, new OpenVPN datasets are generated for heavily mixed traffic with different parameter values. This way we can try to see if this changes the score in such a way, that it will be more in line with the Cisco SSL dataset. The said parameter is the maximum for the random delay after which a new command is executed during the data generation phase. Normally, the heavily mixed traffic is generated with a random delay parameter of between 0 and 1 second. Launching 100 commands with this random delay range would statistically last 50 seconds. In the end, there is an additional 30 seconds given to finish all commands without generating new ones. In figure 5.7 the new datasets are plotted. The number behind OpenVPN represents the parameter for the maximum random delay. As can be seen, with a random delay between 0-3 seconds applied, the same accuracy could be achieved as the Cisco SSL dataset. However, when calculating the Least Squared Error based on the achieved mixing rates for the UDP and SSL datasets, it becomes clear that the value 2 for the delay parameter is most in line with peers. Both options are however larger than the default one, meaning that applications responsible for the network traffic need more time to finish their tasks. As we want to keep the parameters equal across all VPN protocols for fair comparisons, the default parameters are used for the next experiments. Nevertheless, we can conclude that OpenVPN has a lower data throughput resulting in longer execution times per task, leading to higher mixing rates and lower accuracy scores compared to other VPN protocols.

Although this behavior might sound attractive as a measure against fingerprinting, it is questionable how good this will work in real life. As for this research, all data was generated in the same environment, the workings of this possible countermeasure are based on processing power and network speed. Therefore it cannot give any guaranty of how much it will affect the data on different machines, in comparison to padding or minimal delays for example.

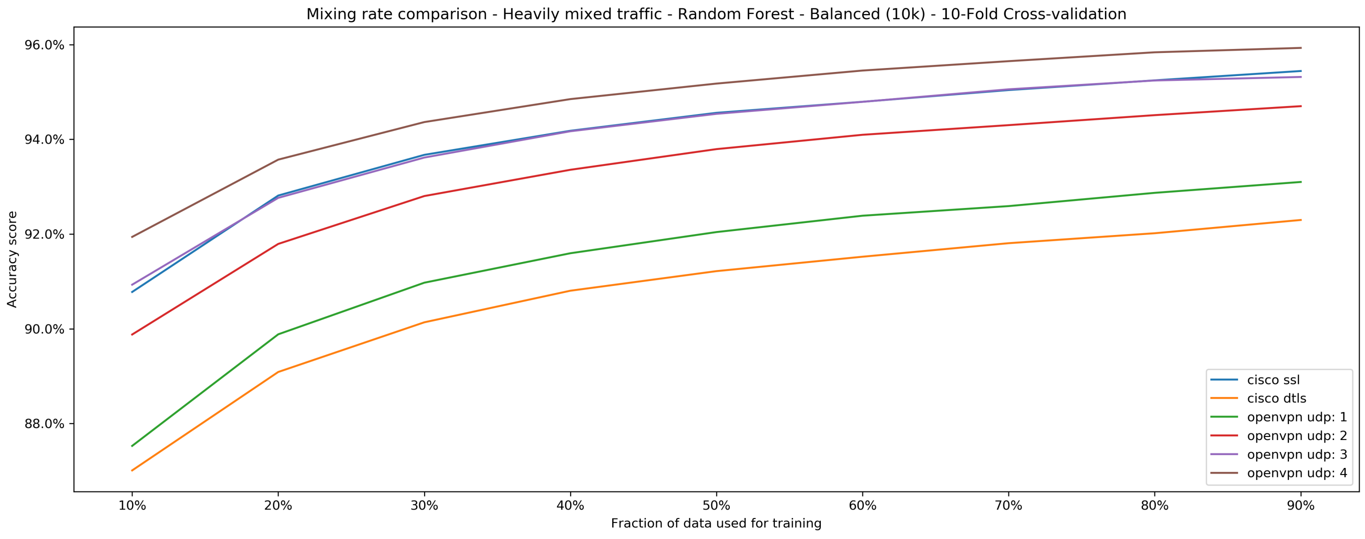


Figure 5.7: Accuracy comparison of different parameters for heavily mixed OpenVPN UDP traffic

5.4.1. OpenVPN TCP

Although OpenVPN TCP showed the same problem as UDP regarding the mixing rate, the difference in accuracy for OpenVPN TCP is too big to be caused only by different mixing rates, so there should be another factor of influence as well. As comes out, it is the only captured VPN protocol that usages Nagle's algorithm for better efficiency. As explained in the background section, the algorithm tries to achieve this by combining several small packets and sending them at once.

Not only do combined packets influence size patterns, but as some packets are buffered for a small time to wait for the next packet, it influences timing patterns as well. Moreover, held packets could distort the order of in and outgoing packets as well. This means that there is no feature which cannot be influenced by Nagle's algorithm as all are based on time, size, and direction. Along with the increased mixing rate, 2 feature distortion factors negatively impact OpenVPN TCP's accuracy scores.

5.5. Network protocol comparison

Now we know how the different VPN protocols score against each other and on which features the classifiers work in general, it is time to take a deeper look into the different network protocols. As the VPN protocols can be effectively grouped based on applying padding or not, we use Cisco SSL and DTLS to represent them as explained in section 5.2. The tables in 5.3 are showing the different scoring metrics between the network protocols. Cisco SSL (which represents the non-padded VPN protocols) are shown in table 5.4a and Cisco DTLS (representing padded protocols) is shown in 5.4b.

Starting to look at Cisco SSL, it comes clear that ICMP and SIP-RTP are the easiest to classify with high precision (low false-positive rate) as well as High recall (low false-negative rate) scores. For ICMP a nearly perfect score is achieved by classifying every ICMP packet as ICMP. These are then followed by HTTP, POP, and IMAP. Although different, all of these protocols contain longer sessions with some standard packet sizes. As HTTP streams often contain multiple MTU packet sizes and acknowledgments, IMAP and POP have to deal with a lot of repetitive commands in-between every email that is requested over the network. This directly explains why SMTP is scoring lower although it has the same purpose as SMTP although the other way around. SMTP setups a new connection for every message instead

(a) Cisco SSL

Protocol	Precision	Recall	F1-score
dns	0.910	0.903	0.907
ftp	0.889	0.888	0.889
http	0.943	0.917	0.930
icmp	0.972	0.994	0.983
imap	0.940	0.930	0.935
pop	0.947	0.925	0.936
sip-rtp	0.970	0.967	0.969
smtp	0.887	0.916	0.902
ssh	0.893	0.910	0.901

(b) Cisco DTLS

Protocol	Precision	Recall	F1-score
dns	0.885	0.871	0.878
ftp	0.818	0.853	0.835
http	0.941	0.917	0.928
icmp	0.912	0.950	0.930
imap	0.922	0.919	0.920
pop	0.914	0.854	0.883
sip-rtp	0.906	0.910	0.908
smtp	0.844	0.857	0.850
ssh	0.890	0.895	0.893

Table 5.3: Performance of heavily mixed traffic per network protocol

of sending a single repetitive command multiple times to let the server know it has more messages to send. SMTP comes together both DNS and SSH as last. This is likely caused by the fact that there are less strict size or time patterns observed during a session. While DNS and SMTP sessions are often really short, SSH can vary more in length, although it is highly dependent on user input.

Looking at the DTLS table, it is interesting to see that the order of protocols has shifted. Where ICMP and SIP-RTP were on top for SSL they are around HTTP etc. now. Since the only difference between SSL and DTLS is the usage of rounded packet sizes, the reason behind this should be in line with that fact. When inspecting the most frequent packet sizes for ICMP and SIP-RTP, it comes clear that for both protocols, the most common packet sizes end up in the largest and third-largest size frequencies, making it much harder to train classifiers for their unique sizes. This also explains why the scores for HTTP did not drop, as many of the HTTP packets already have the maximum packet size, which is fixed for SSL as well as DTLS.

Although padded VPN protocols make it harder to correctly classify network protocols, it might be useful to predetermine which traffic you want to hide. This way you can extend or shift the padding in such a way that most packets will end up in the popular size frequencies, making the padding most effective. Moreover, user randomness by input could help to break timing patterns. Although sending emails is user input as well, that single action causes a chain of network packets to adhere to a specific protocol pattern, whereas SSH only sends the minimum packets needed to transfer the commands or output.

5.6. Markov influence

As described in the methodology, both random forest, as well as Markov classifiers, are used during this research. They could be used as a stand-alone classifier or the likeliness scores of the Markov classifiers could be used as a feature for the random forests. Although the Markov chain is used for calculating likeliness scores of state transitions occurring, a scoring method should be picked first. This method was applied in section 5.2 to enrich the random forest classifier.

As the likeliness scores could be calculated in different ways by using a different number of state transitions, the best method should be determined. Based on the available features for the whole dataset, 4 different scoring methods are created. As all standard features are looking back and forth at a maximum of 2 packets, this will be the range for the scoring as well. The following 4 methods were compared: 3-Markov (multiplying transitions 1 back and 1 forth), 5-Markov (2 back, 2 forth), Markov-ML (maximum l likelihood 1 back, 1 forth), Markov-ML3 (multiplying highest transition back and forth).

Table 5.5 (appendix A.6 and A.7) shows the accuracy scores for the different decision algorithms. Although not as good as the random forests, the Markov classifiers alone could still classify the different protocols achieving around 55% for non-padded and 43% for padded protocols with heavily mixed traffic. Moreover, it could be concluded that the maximum-

VPN Protocol	3-Markov	5-Markov	Markov-ML	markov-ML3
cisco_ssl	0.550	0.560	0.460	0.460
fortigate_ssl	0.570	0.570	0.470	0.480
openvpn_udp	0.470	0.440	0.410	0.380
cisco_dtls	0.430	0.460	0.360	0.400
wireguard	0.430	0.440	0.340	0.390
ipsec	0.410	0.400	0.330	0.350
pulse_ipsec	0.430	0.430	0.350	0.390
openvpn_tcp	0.440	0.400	0.390	0.330

Table 5.5: Markov classifier accuracy comparison for heavily dataset using the matrix method

likelihood methods perform worse than the methods which are taking all packets into account. An advantage of Markov Chains above random forests is the fact that there is no need for a training phase. Only the model should have enough input data to create a trustworthy state representation.

It might already be clear that the accuracy scores of the Markov Chains are lower than those from the random forests as seen in the previous sections. Table 5.6 compares the scores between random forests which include Markov chain outputs as features and those which don't. The differences in scores are really small, which is caused by the fact that the own accuracy scores from the Markov classifiers were way lower than the base score of the random forest, meaning it is more capable on its own to find patterns. Although the random forests are achieving higher scores, they still can be used as a quick classifier for new VPN or network protocols once the Markov model was created. Besides it shows that even simpler models can see patterns of network protocols in busy VPN environments, emphasizing once again the existing leakage model. In that sense, Markov models are a fast solution to quickly identify if network protocols show underlying patterns, without the need to train more advanced classifiers.

VPN Protocol	No Markov	Markov included
cisco_ssl	0.927	0.928
fortigate_ssl	0.929	0.929
openvpn_udp	0.909	0.911
cisco_dtls	0.893	0.893
wireguard	0.880	0.879
ipsec	0.882	0.883
pulse_ipsec	0.876	0.878
openvpn_tcp	0.825	0.823

Table 5.6: Accuracy comparison of the effect if including Markov features for heavily mixed traffic

5.7. Countermeasures

Since VPNs are used for various reasons, not limited to bypassing geographically blocked content but including safety and freedom of speech, it is important to explore where possibilities are to implement countermeasures. During this research, we found out that the ability to fingerprint network protocols is based on underlying patterns of 3 features (namely direction, size, and timing). A logical starting point for researching countermeasures would be based on the ability to hide them. Since most obfuscations are not developed, and only optimized solutions won't affect other features as well, we study the countermeasures by applying the obfuscation technique to the parsed dataset. Moreover, this gives us the possibility to study (one-way) client obfuscations versus (bidirectional) protocol obfuscations as well. The used measures are compared based on the effectiveness of normal-sized and padded-sized VPN protocols. Metrics are based on the accuracy scores, where more research and actual implementations are needed to concretely discuss the impact on the performance of the VPNs themselves in terms of speed and usability.

5.7.1. Size Obfuscations

The size obfuscations are based on the ones found in the literature, namely MTU padding, mice-elephant, exponential, and multiplication (rounding up to nearest 128 bytes). Moreover, we could use a distribution mapping of the size feature to pad in such a way that we can achieve homogeneity scores of 25, 50, or 75% percent (where 100% homogeneity is equal to MTU padding). The graphs for SSL and DTLS traffic (5.8, 5.9) show the effects of the applied obfuscation techniques on heavily mixed traffic. As can be seen, the best obfuscation, full MTU padding, is only capable of dropping the accuracy to 60%, which is not even near close to the guessing rate. Other interesting findings are the fact that mice-elephant is scoring comparable to exponential but way less than multiplication-128 padding. This is interesting as mice-elephant only has 2 different sizes whereas the others have 8 to 10 size options. However, this could be explained by the fact that the minimum size of mult-128 padding is 128 bytes, whereas the is higher than 75% of all packet sizes. This is much higher than the minimum for exponential is 64 (as this is the first size higher than a normal packet size), or the value for the mice (which was picked to be the value covering 50% of the size distribution). As the lower size distribution is quite dense, combining all of them to a single packet size is more effective than a split in the middle.

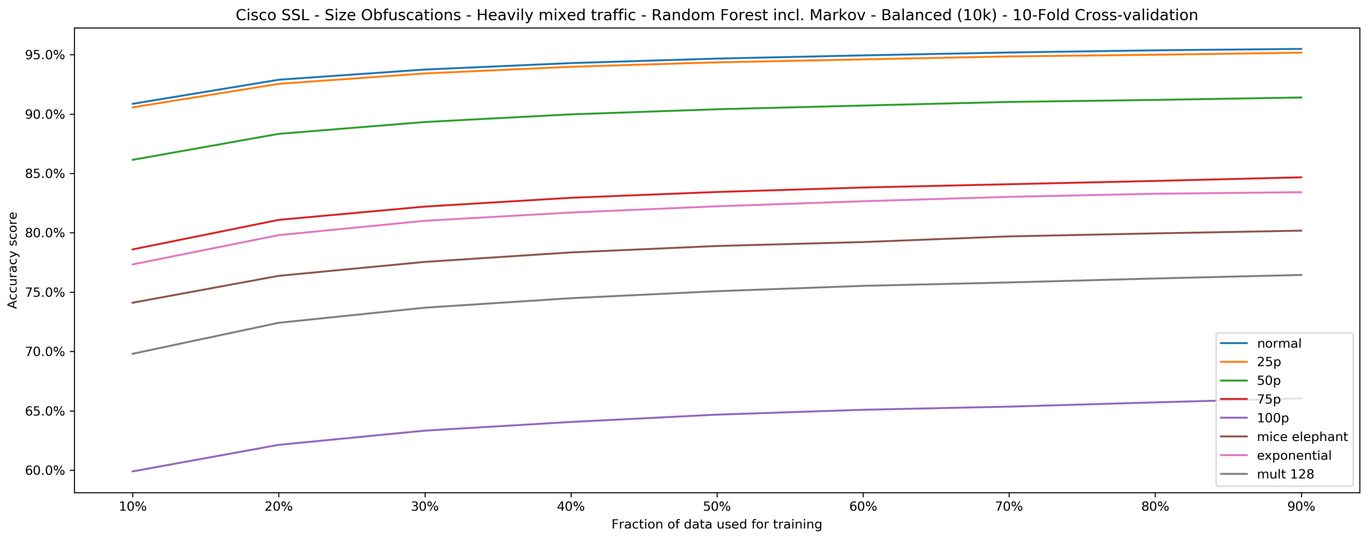


Figure 5.8: Accuracy comparison with different size obfuscations applied for Cisco SSL heavily mixed traffic

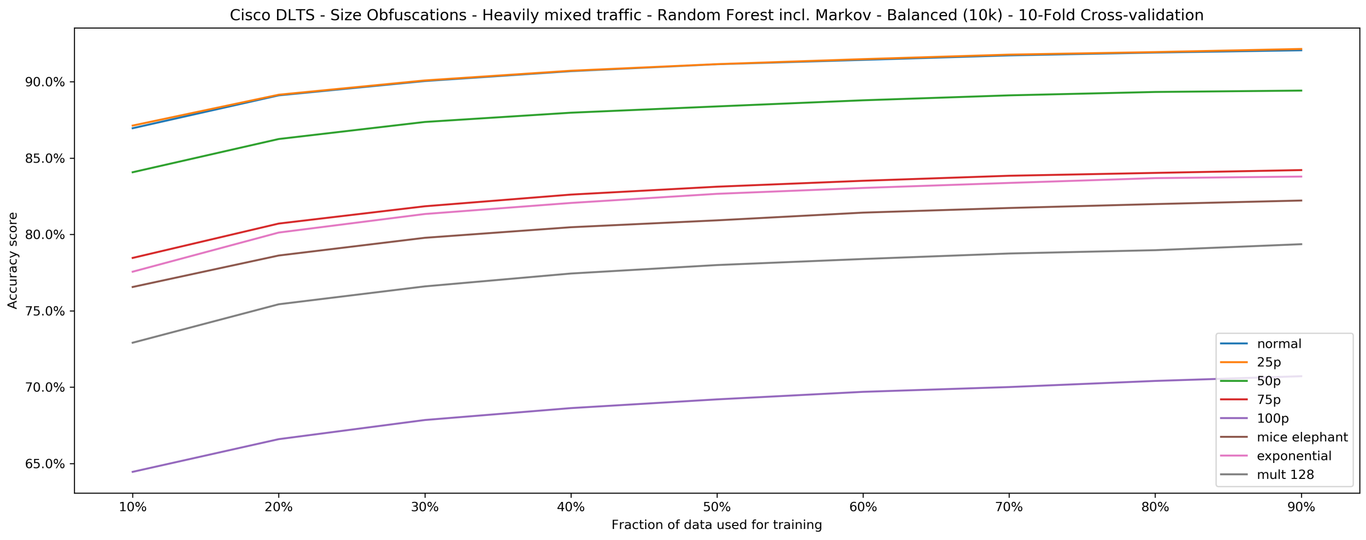


Figure 5.9: Accuracy comparison with different size obfuscations applied for Cisco DTLS heavily mixed traffic

As discussed in section 5.3, by temporarily removing all size and burst detection features from the feature set, it was shown that the timing features are still good enough to acquire an accuracy score of 60% 5.5, which is in line with the full MTU padding scores. To compare the real influence of the obfuscations on the characteristics it tries to break, the performance metrics are included of both the full feature set as well as the scores for the selective feature set based on which feature the method tries to influence. In appendix A.4 all the different tables which different options are fully shown. Here it is shown that only full MTU padding is capable of reaching the guessing rate, where the second-best (mult-128) is only reaching 50% accuracy. On the other hand, the accuracy scores do increase when the obfuscations are only applied on client-side traffic.

Concluding on size obfuscations, it becomes clear that full MTU padding works best. There is a large difference between that and the second-best performing obfuscation, namely linear (multiplication) padding. Moreover, the accuracy scores for multiple obfuscations are closer together for the padded protocols. Since there are fewer unique packet size values for them, the size feature already contained less information. As the size obfuscations tend to do the same, it is logical that the impact of different methods is closer together as the amount of entropy removal is lower. Applying size obfuscation in the real world, however, it is hard to argue which obfuscation methods are feasible, as they have a large overhead. This could lead to network infrastructure limitations and delays for clients using them.

5.7.2. Time Obfuscations

In terms of time obfuscations, there are multiple options. First of all, we could try to achieve homogeneity of minimum delays based on the timing distributions observed. Secondly, we could apply a random time delay based on different distributions like a Gaussian or a Pareto. When comparing all the different timing obfuscations as shown in figures 5.10 and 5.11, the first thing that comes clear immediately is the fact the effectiveness of timing obfuscations is way less than size obfuscations reaching a minimum accuracy of 80% for randomly applied delays. This is interesting since distorting the feature is working better than effectively removing it by having the exact same delay for every data packet. Moreover, although the differences are minimal, applying the delays per direction seem to work slightly better then applying it to the next packet unrelated to the direction.

As shown before, size is the most important factor where timing comes second. Besides it could be argued that due to the high bandwidth speeds nowadays, the effects of size obfuscations are less noticeable to the end-user than time obfuscations since that increases the delay significantly. Nevertheless, it should be taken into account that applying size obfuscations on a large scale could affect everyone by the increased network traffic.

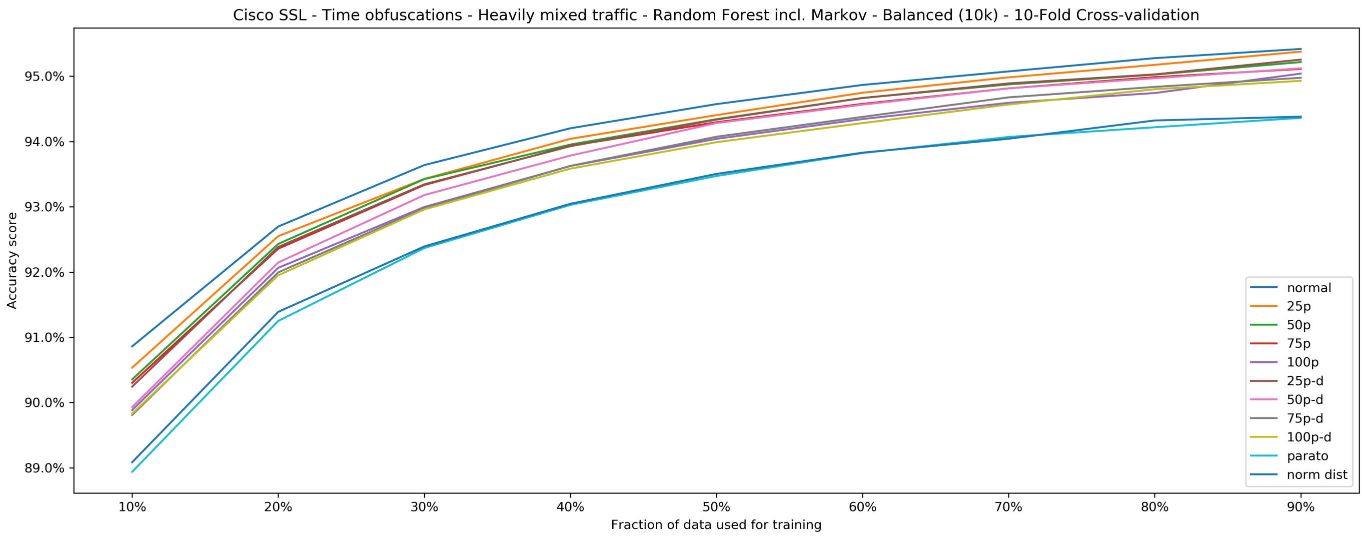


Figure 5.10: Accuracy comparison with different time obfuscations applied for Cisco SSL heavily mixed traffic

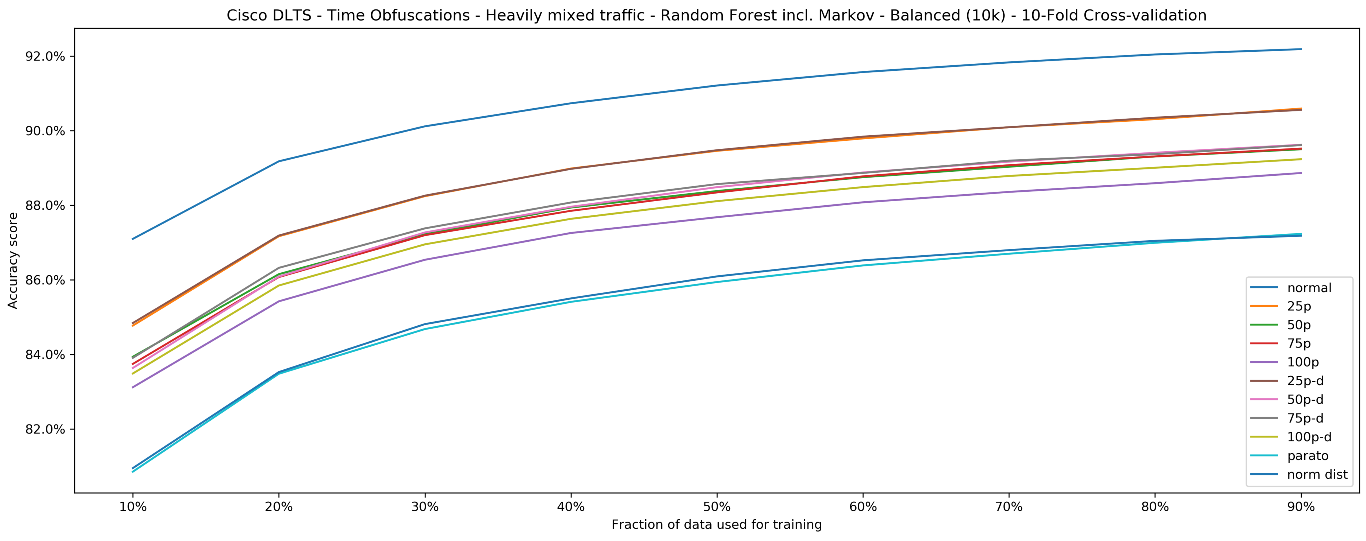


Figure 5.11: Accuracy comparison with different time obfuscations applied for Cisco DTLS heavily mixed traffic

5.7.3. Applying obfuscation in real-world situations

The most important takeaway from these obfuscation experiments is the fact that size obfuscations alone are not enough when the number of output classes is not that high and each of the classes shows distinctive patterns in terms of size and timing. During the earlier experiments, Nagle's algorithm was observed for OpenVPN TCP. As this is combining multiple packets into one both packet timing as ordering as distorted. Including background noise, it could prevent situations where only 1 protocol is being used at the same time, basically the difference between single and heavily mixed traffic. Nevertheless, we could conclude that having only 1 obfuscation method at the same time would probably not hide enough patterns, and more research is needed to find the ideal combination in terms of effectiveness and resource efficiency.

On a final note, one should consider that probably not all of the above measures are applied to both traffic directions. It could be argued that not all VPN solutions would include such obfuscations methods and it is, therefore, interesting to see how the obfuscation measures behave when only applied to the client-server communication but not to the other way around. The results can be found in Appendix A.4 split into multiple tables per VPN protocol and obfuscation feature. Shortly said, applying full MTU padding on client-side traffic without timing features still has a performance of 60% where it seems that server packet sizes combined with in- and outgoing patterns are more important than client packet sizes. Unfortunately, this means that hiding your metadata won't be that easy without an obfuscation technique implemented by the VPN provider in the software, which is another requirement to think of when new methods are going to be designed.

6 Conclusion

In this thesis, we investigated the possibilities of network protocol fingerprinting while they are sent over a virtual private network tunnel. In comparison to related work, the VPN introduces a new important factor, namely the fact that a single channel is used for all streams making it impossible to split each flow separately. We show that, although working with totally different features based on consecutive packets rather than stream statistics, it is possible to determine the network protocols with relatively high certainty.

As related research only used datasets with 1 network protocol per session, we had to capture our own data. We picked 9 common network protocols used for day-to-day tasks, automated the generation process for 3 different levels of inter-protocol usage, and generated the data for a total of 8 different VPN protocols. The different mixing rates allowed us to compare the characteristics of a VPN when activity on the host differs while the latter was used to compare the influence of characteristics that differs between different VPN protocols. As implementing different obfuscations methods for different VPN protocols is hard and time-consuming to do correctly, these are studied by applying the effects on the parsed data to simulate the expected behavior.

6.1. Research questions

The experiments of this thesis were shaped to fill the research gap of VPN fingerprinting. The main research question is: *To which extent do VPN-protocols, -solutions and -obfuscations differ when comparing the fingerprintability of different network protocols used inside a VPN-tunnel?* To answer this fully, multiple sub-questions were formed as well, all being answered with different parts of the experiments, allowing us to conclude on the main question in the end.

6.1.1. Sub-questions

Which effect has the mixing of protocols over an encrypted stream on the scoring metrics? Starting with the non-mixed traffic, scores of 99% accuracy are reached for all different VPN protocols, something that is in line with earlier conducted research taking non-mixed traffic into account. For network protocol fingerprinting we could say that a VPN does protect your privacy as well as wanted in an environment free of interference and noise. Increasing that first condition by mixing different tasks at the same time, the scores begin to drop to 97% while the variance increases. Further increasing the mixing rate to have at least 2 different protocols at each time interval for 70% of the time, lowers the accuracy to an average of around 88%. With the limited set of 9 network protocols being used, it is possible to fingerprint them with relatively high accuracy. This shows that network protocol fingerprinting is indeed possible. Future work should examine how effective this method is when applied to real-world situations. Given that the number of network protocols will be higher and protocol can be used in multiple ways, for example when using the secure version of some protocols, it is interesting to find out how scores would evolve.

How do different VPN protocols and solutions score amongst each other? The selected VPN protocols can be categorized in multiple ways. Padded vs non-padded, cloud vs on-premise, and open source vs enterprise. It is shown that only padded/non-padded protocols show a clear distinction. OpenVPN not taken into consideration, one could say that non-padded protocols score around 5% point higher than the padded ones for heavily mixed traffic. As the padded protocols only influence the size future, picking a padded VPN protocol is a better choice when one wants to lower the fingerprintability. Looking at the cloud vs on-premise hosted VPNs, no significant score changes are observed. This is somewhat unexpected as the placement of a VPN could influence timing features as longer routes often introduce more and unexpected delays. However, as we didn't compare different placements of the capturing devices along the route, it could be the case that having a sniffer in the middle would observe lower scores due to timing problems. Lastly, the scores between open-source and enterprise VPNs do not indicate that enterprise VPNs offer better fingerprint protection by default. Although some have more extensive configurable options, no settings were used by default to prevent this attack.

Only both OpenVPN protocols were (a little bit) misaligned with the other VPN protocols. Inspecting the mixing rates of the generated traffic showed a higher ratio, although generated with the same settings. When changing the generator parameters to end up with the same mixing rates as for the other VPN protocols, the accuracy scores increased as well, being more in line with the peers. AS this solved the misalignment for OpenVPN-UDP fully, this was not the case for OpenVPN-TCP. The last part could be explained by the fact it uses Nagle's algorithm for more efficient bandwidth usage. By buffering and combining several smaller packages, the timing and patterns features were distorted. Although OpenVPN might be slower in usage, which explains the higher processing time of network packets and thus higher mixing rates, it forms a natural mitigation method to lower the fingerprintability of the network protocols.

Which data features are most important for the fingerprintability of VPN traffic? The more the protocols become mixed, the less important timing becomes. The size features are in all cases the most important one, although only for 10% for non-mixed traffic, growing toward 25% for heavily mixed. These percentages are valid for non-padded protocols, as the size importance dropped a few percentages when a VPN applies padding. By classifying time and size features separately, it was proven that this was caused by the lower information rate of the size feature rather than better information hidden in the timings. This also implies that the location of the VPN and packet collector is of less influence than expected in the beginning since the timing features are being of lower importance compared to size.

To which extent are some network protocols easier to fingerprint than others? Although it is hard to conclude with a hard order, since the score differences are so small for some protocols, some conclusions can be drawn. ICMP and SIP-RTP are scoring the highest followed by a group of 3 protocols namely HTTP, IMAP, and POP. Thereafter comes DNS ending with another group of 3 consisting of FTP, SMTP, and SSH. This can be explained by the underlying protocol patterns. Both ICMP and SIP-RTP are based on a constant rate of packets of the same size. Where HTTP has the longest streams with MTU sizes compared to the other protocols, POP and IMAP are both passed on repetitive patterns of commands in-between every mail. SMTP however sets up a new connection for every mail. The same goes for DNS that is often based on short sessions with variable packet lengths. Since all classifications are done with a training set containing the same amount of packets of protocols, minority classes are not a problem. However, using this attack on real-life data one should be aware that some protocols use occur less often or with a lower number of packets, making it harder to train on those without some additional filtering.

Which mitigations can be taken to protect our data better and how do they good do they perform? Divided into two main categories, size and time, different obfuscations are tested. In general, it could be concluded that size obfuscations work better, being able to lower the accuracy towards 60% when applying full MTU padding. The time obfuscations were only able to lower the score to 80%. Both scores are in line with the earlier results from classifying with size or time features only. All combined, it means that there is still a large amount of information left to be used for classifying when only obfuscation one of the 2 main categories. Combining obfuscation methods seems to be necessary to fully hide underlying patterns but this comes with huge overhead costs. Lastly, reviewing the VPN protocols shows us another possible obfuscation. Nagle's algorithm, as seen by OpenVPN-TCP, also helps to lower the fingerprintability by buffering and combining packets, distorting packet patterns and timing. And, it is expected that the accuracy scores will drop in the real world anyway, as users use a higher number of different protocols, and telemetry and other noises are added in the background, making the whole classification a lot harder.

6.1.2. Main question

To which extent do VPN-protocols, -solutions and -obfuscations differ when comparing the fingerprintability of different network protocols used inside a VPN-tunnel? During this study, various VPN protocols are compared to whether or not incorporated into enterprise solutions. In a closed world setting, we were able to classify 9 different network protocols used in both quiet and busy channels. A near-perfect score was obtained on single traffic streams, which dropped towards 82% as a minimum when classifying OpenVPN-TCP traffic in the busiest traffic setting generated. Overall it could be said that there is not a large difference between VPN products where opensource and enterprise solutions scored similarly. The only fundamental difference was observed between padded and nonpadded protocols as the latter scored around 5% point lower in busy channels. The only VPN protocol that stood out was OpenVPN. Both UDP and TCP variants scored lower, although that was caused by having a higher mixing rate and the usage of Nagle's algorithm. Classifying OpenVPN traffic generated with similar mixing rates and having Nagle's algorithm disabled for TCP (which both are parameters that are not directly linked to the protocol), results in similar scores as other protocols.

In terms of feature importance, it seems that size features are more important than time features when classifying packets individually instead of streaming-wise. This was observed for obfuscations as well, where size obfuscations performed way better than timing mitigations. In this closed world setting the timing features were rich enough to prevent a near guessing rate score when size features disappear. However, this will most likely drop when telemetry and other noises are added. Therefore it is likely that size obfuscations would solve most of the fingerprinting issues in real-world settings although it comes with some overhead.

6.2. Limitations

As mentioned many times before, this study was done by generating data in a closed world environment. Although the servers were publicly hosted, the traffic was generated inside a docker container to limit noises and increase reproducibility. Furthermore, only 9 network protocols were compared, all quite different. The commands were generated separately and automated via the command line. Normal desktop applications are more likely to combine different protocols for more user convenience, like getting HTML content from servers when opening a mail, for example, resulting in different packet streams.

Secondly, due to a large number of different factors, this study was limited to only using random forest and Markov classifiers. These classifiers were amongst the best performing classifiers in literature. Other used models in literature were mostly black-box models, meaning we couldn't use them to extract the feature importance. However, it could be that with a larger number of network protocols, the random forest model might not be the best classifier available. When feature importance is not needed, other classifiers should be reconsidered, as neural networks were achieving high scores in literature as well.

Thirdly the obfuscations methods were applied after generating the network traffic. Effects of padding on timing features due to larger packet sizes were not taken into account. Besides, it could potentially influence packet ordering when obfuscations are only applied on one side. So performance scores with obfuscations applied are best-case situations.

6.3. Future work

During this research, it was shown that fingerprinting VPNs is certainly possible. However, large-scale experiments could show the real impact on society as a lot of factors might change. This could show whether or not this is a realistic scenario of privacy infringements that could be used by for example repressive governments. As the size of the protocol set, noise and jitter, and the amount of traffic all increase, new problems may arise that should be tackled. Another interesting field would be to test and apply existing and new obfuscation methods. When the attack is still realistic to apply on a larger scale, the need for protection rises as well.

A Tables

A.1. Classification result tables

VPN Protocol	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
cisco_ssl	0.995	0.995	0.995	0.995
fortigate_ssl	0.995	0.995	0.995	0.995
openvpn_udp	0.993	0.993	0.993	0.993
cisco_dtls	0.995	0.995	0.995	0.995
wireguard	0.993	0.993	0.993	0.993
ipsec	0.992	0.992	0.992	0.992
pulse_ipsec	0.993	0.993	0.993	0.993
openvpn_tcp	0.993	0.993	0.993	0.993

Table A.1: Performance of single mixed traffic per protocol

VPN Protocol	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
cisco_ssl	0.978	0.978	0.978	0.978
fortigate_ssl	0.976	0.977	0.976	0.976
openvpn_udp	0.968	0.968	0.968	0.968
cisco_dtls	0.970	0.970	0.970	0.970
wireguard	0.966	0.966	0.966	0.966
ipsec	0.968	0.969	0.968	0.968
pulse_ipsec	0.966	0.966	0.966	0.966
openvpn_tcp	0.944	0.944	0.944	0.944

Table A.2: Performance of partially mixed traffic per protocol

VPN Protocol	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
cisco_ssl	0.927	0.928	0.927	0.927
fortigate_ssl	0.929	0.929	0.929	0.929
openvpn_udp	0.911	0.911	0.911	0.911
cisco_dtls	0.892	0.893	0.892	0.892
wireguard	0.878	0.879	0.878	0.878
ipsec	0.882	0.883	0.882	0.882
pulse_ipsec	0.876	0.878	0.876	0.877
openvpn_tcp	0.822	0.823	0.822	0.822

Table A.3: Performance of heavily mixed traffic per protocol

A.2. Network protocol comparison

Predicted Protocol Actual Protocol	dns	ftp	http	icmp	imap	pop	sip-rtp	smtp	ssh
dns	72512	1663	1006	580	847	458	1029	800	1105
ftp	2369	70972	722	245	334	1449	193	2227	1489
http	734	928	73256	3	944	556	56	1294	2229
icmp	78	34	39	79423	307	2	1	87	29
imap	1154	502	313	746	74371	631	50	1242	991
pop	669	2614	333	57	894	73459	101	1152	721
sip-rtp	390	216	245	78	858	235	77199	136	643
smtp	868	1985	524	496	543	511	140	73463	1470
ssh	858	1219	1355	32	318	469	901	2010	72838

Table A.4: Cross tabulation of heavily mixed traffic for Cisco SSl per network protocol

Predicted Protocol Actual Protocol	dns	ftp	http	icmp	imap	pop	sip-rtp	smtp	ssh
dns	69539	1547	1359	2383	1082	390	1548	954	1198
ftp	2371	68147	635	792	189	1725	1576	3266	1299
http	811	834	73335	4	844	464	110	1330	2268
icmp	604	680	20	75935	1360	105	1	1018	277
imap	1213	335	252	1576	73074	825	77	1576	1072
pop	694	4531	212	580	741	68242	2223	2146	631
sip-rtp	1621	993	337	84	759	1580	73221	817	588
smtp	837	5030	446	1425	502	812	748	68714	1486
ssh	1130	1278	1436	288	380	535	1340	2212	71401

Table A.5: Cross tabulation of heavily mixed traffic for Cisco DTLS per network protocol

A.3. Markov model classification scores

VPN Protocol	3-Markov	5-Markov	Markov-ML	markov-ML3
cisco_ssl	0.790	0.910	0.640	0.810
fortigate_ssl	0.790	0.900	0.630	0.810
openvpn_udp	0.790	0.910	0.660	0.820
cisco_dtls	0.630	0.750	0.520	0.680
wireguard	0.610	0.730	0.520	0.660
ipsec	0.640	0.760	0.490	0.670
pulse_ipsec	0.630	0.760	0.500	0.670
openvpn_tcp	0.770	0.890	0.640	0.810

Table A.6: Markov classifier accuracy comparison for single dataset using the matrix method

VPN Protocol	3-Markov	5-Markov	Markov-ML	markov-ML3
cisco_ssl	0.690	0.780	0.560	0.680
fortigate_ssl	0.730	0.790	0.580	0.720
openvpn_udp	0.690	0.770	0.590	0.680
cisco_dtls	0.510	0.600	0.460	0.540
wireguard	0.510	0.580	0.410	0.540
ipsec	0.530	0.590	0.440	0.540
pulse_ipsec	0.500	0.570	0.410	0.520
openvpn_tcp	0.660	0.710	0.550	0.620

Table A.7: Markov classifier accuracy comparison for partially dataset using the matrix method

A.4. Counter measures performance tables

A.4.1. Size obfuscation results

SSL

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.928	0.928	0.928	0.928
25p	0.925	0.925	0.925	0.925
50p	0.884	0.886	0.884	0.884
75p	0.809	0.812	0.809	0.810
100p	0.621	0.624	0.621	0.622
mice_elephant	0.768	0.768	0.768	0.767
exponential	0.797	0.800	0.797	0.798
mult_128	0.725	0.733	0.725	0.728

Table A.8: Accuracy scores of Cisco SSL size obfuscations

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.928	0.928	0.928	0.928
25p	0.926	0.927	0.926	0.926
50p	0.910	0.911	0.910	0.911
75p	0.880	0.881	0.880	0.880
100p	0.856	0.857	0.856	0.856
mice_elephant	0.893	0.894	0.893	0.893
exponential	0.885	0.886	0.885	0.885
mult_128	0.869	0.870	0.869	0.869

Table A.9: Accuracy scores of Cisco SSL size obfuscations client-side

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.929	0.929	0.929	0.929
25p	0.924	0.924	0.924	0.924
50p	0.830	0.835	0.830	0.832
75p	0.665	0.686	0.665	0.671
100p	0.176	0.081	0.176	0.100
mice_elephant	0.535	0.532	0.535	0.532
exponential	0.669	0.673	0.669	0.670
mult_128	0.474	0.495	0.474	0.473

Table A.10: Accuracy scores of Cisco SSL size obfuscations selected features

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.930	0.930	0.930	0.930
25p	0.926	0.926	0.926	0.926
50p	0.897	0.897	0.897	0.897
75p	0.848	0.849	0.848	0.848
100p	0.829	0.830	0.829	0.830
mice_elephant	0.869	0.869	0.869	0.869
exponential	0.871	0.872	0.871	0.871
mult_128	0.840	0.840	0.840	0.840

Table A.11: Accuracy scores of Cisco SSL size obfuscations client-side selected features

DTLS

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.890	0.891	0.890	0.890
25p	0.891	0.892	0.891	0.891
50p	0.865	0.867	0.865	0.866
75p	0.807	0.811	0.807	0.808
100p	0.667	0.671	0.667	0.668
mice_elephant	0.786	0.787	0.786	0.786
exponential	0.801	0.803	0.801	0.801
mult_128	0.752	0.759	0.752	0.754

Table A.12: Accuracy scores of Cisco DTLS size obfuscations

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.892	0.892	0.892	0.892
25p	0.892	0.893	0.892	0.892
50p	0.879	0.880	0.879	0.879
75p	0.851	0.852	0.851	0.851
100p	0.826	0.828	0.826	0.827
mice_elephant	0.861	0.862	0.861	0.861
exponential	0.858	0.859	0.858	0.858
mult_128	0.839	0.841	0.839	0.840

Table A.13: Accuracy scores of Cisco DTLS size obfuscations client-side

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.844	0.844	0.844	0.843
25p	0.842	0.842	0.842	0.841
50p	0.763	0.766	0.763	0.763
75p	0.609	0.636	0.609	0.616
100p	0.175	0.081	0.175	0.106
mice_elephant	0.546	0.545	0.546	0.544
exponential	0.645	0.651	0.645	0.647
mult_128	0.483	0.509	0.483	0.478

Table A.14: Accuracy scores of Cisco DTLS size obfuscations selected features

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.842	0.842	0.842	0.842
25p	0.841	0.841	0.841	0.841
50p	0.809	0.810	0.809	0.809
75p	0.752	0.756	0.752	0.753
100p	0.717	0.719	0.717	0.717
mice_elephant	0.778	0.779	0.778	0.778
exponential	0.789	0.790	0.789	0.789
mult_128	0.739	0.740	0.739	0.739

Table A.15: Accuracy scores of Cisco DTLS size obfuscations client-side selected features

A.4.2. Time obfuscation results

SSL

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.928	0.928	0.928	0.928
25p	0.926	0.926	0.926	0.926
50p	0.924	0.924	0.924	0.924
75p	0.925	0.925	0.925	0.925
100p	0.921	0.921	0.921	0.921
25p-d	0.924	0.924	0.924	0.924
50p-d	0.921	0.921	0.921	0.921
75p-d	0.920	0.920	0.920	0.920
100p-d	0.919	0.919	0.919	0.919
parato	0.913	0.913	0.913	0.913
norm_dist	0.912	0.912	0.912	0.912

Table A.16: Accuracy scores of Cisco SSL time obfuscations

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.928	0.928	0.928	0.928
25p	0.927	0.927	0.927	0.926
50p	0.927	0.927	0.927	0.927
75p	0.927	0.927	0.927	0.927
100p	0.927	0.927	0.927	0.927
25p-d	0.925	0.925	0.925	0.925
50p-d	0.926	0.926	0.926	0.926
75p-d	0.926	0.926	0.926	0.926
100p-d	0.926	0.926	0.926	0.926
parato	0.920	0.920	0.920	0.920
norm_dist	0.922	0.922	0.922	0.921

Table A.17: Accuracy scores of Cisco SSL time obfuscations client-side

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.620	0.622	0.620	0.620
25p	0.489	0.494	0.489	0.491
50p	0.415	0.419	0.415	0.416
75p	0.387	0.390	0.387	0.385
100p	0.310	0.303	0.310	0.300
25p-d	0.523	0.523	0.523	0.523
50p-d	0.469	0.469	0.469	0.469
75p-d	0.450	0.449	0.450	0.449
100p-d	0.404	0.402	0.404	0.403
parato	0.211	0.202	0.211	0.204
norm_dist	0.203	0.193	0.203	0.196

Table A.18: Accuracy scores of Cisco SSL time obfuscations selected features

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.616	0.619	0.616	0.617
25p	0.569	0.571	0.569	0.570
50p	0.558	0.559	0.558	0.558
75p	0.551	0.553	0.551	0.552
100p	0.534	0.535	0.534	0.534
25p-d	0.562	0.563	0.562	0.562
50p-d	0.549	0.550	0.549	0.549
75p-d	0.530	0.531	0.530	0.531
100p-d	0.501	0.502	0.501	0.501
parato	0.413	0.411	0.413	0.411
norm_dist	0.418	0.416	0.418	0.416

Table A.19: Accuracy scores of Cisco SSL time obfuscations client-side selected features

DTLS

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.890	0.891	0.890	0.891
25p	0.870	0.871	0.870	0.870
50p	0.862	0.862	0.862	0.862
75p	0.859	0.859	0.859	0.859
100p	0.856	0.856	0.856	0.855
25p-d	0.872	0.872	0.872	0.872
50p-d	0.862	0.863	0.862	0.862
75p-d	0.862	0.862	0.862	0.862
100p-d	0.858	0.858	0.858	0.858
parato	0.835	0.836	0.835	0.835
norm_dist	0.836	0.836	0.836	0.835

Table A.20: Accuracy scores of Cisco DTLS time obfuscations

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.891	0.891	0.891	0.891
25p	0.875	0.876	0.875	0.875
50p	0.874	0.875	0.874	0.875
75p	0.875	0.875	0.875	0.875
100p	0.872	0.872	0.872	0.872
25p-d	0.874	0.875	0.874	0.874
50p-d	0.871	0.872	0.871	0.871
75p-d	0.871	0.872	0.871	0.871
100p-d	0.870	0.871	0.870	0.870
parato	0.854	0.855	0.854	0.854
norm_dist	0.856	0.856	0.856	0.856

Table A.21: Accuracy scores of Cisco DTLS time obfuscations client-side

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.664	0.668	0.664	0.665
25p	0.539	0.546	0.539	0.541
50p	0.394	0.397	0.394	0.393
75p	0.384	0.385	0.384	0.380
100p	0.312	0.306	0.312	0.304
25p-d	0.560	0.562	0.560	0.561
50p-d	0.480	0.480	0.480	0.480
75p-d	0.464	0.463	0.464	0.463
100p-d	0.422	0.421	0.422	0.421
parato	0.211	0.202	0.211	0.204
norm_dist	0.204	0.194	0.204	0.196

Table A.22: Accuracy scores of Cisco DTLS time obfuscations selected features

Obfuscation method	Accuracy	Macro-average Precision	Macro-average Recall	Macro-average F1-score
normal	0.663	0.666	0.663	0.664
25p	0.604	0.607	0.604	0.605
50p	0.591	0.593	0.591	0.592
75p	0.592	0.594	0.592	0.592
100p	0.577	0.579	0.577	0.578
25p-d	0.599	0.602	0.599	0.600
50p-d	0.582	0.584	0.582	0.583
75p-d	0.571	0.573	0.571	0.572
100p-d	0.544	0.546	0.544	0.545
parato	0.453	0.453	0.453	0.453
norm_dist	0.455	0.456	0.455	0.455

Table A.23: Accuracy scores of Cisco DTLS time obfuscations client-side selected features

B
Figures

B.1. Classification result figures

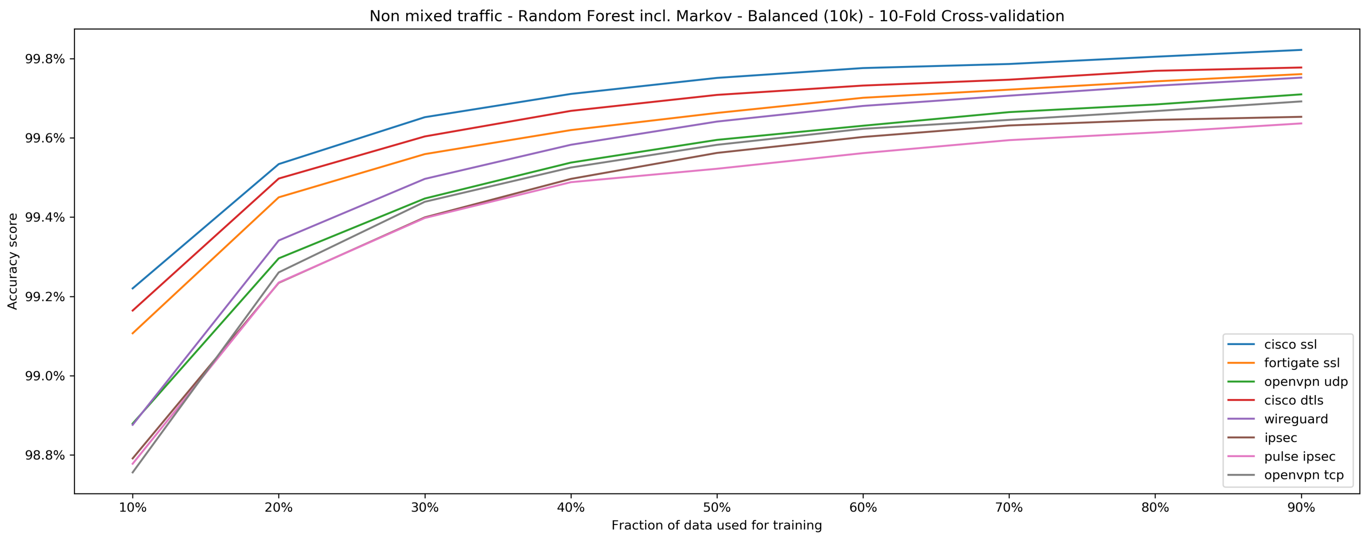


Figure B.1: Accuracy scores for non mixed traffic

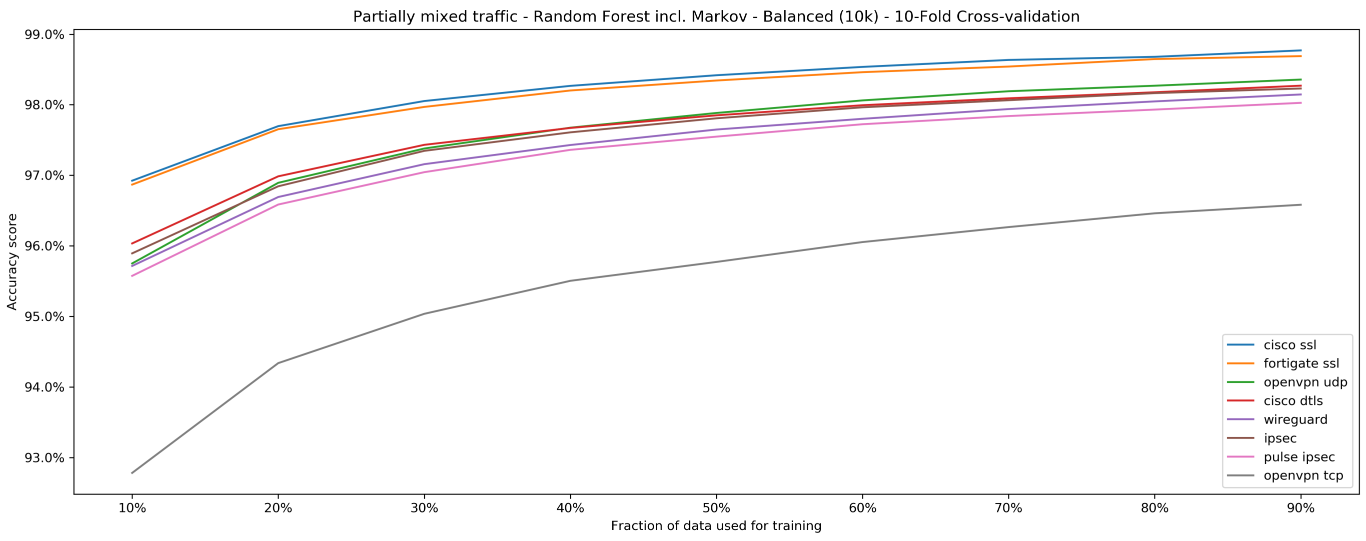


Figure B.2: Accuracy scores for partially mixed traffic

B.2. Feature comparison

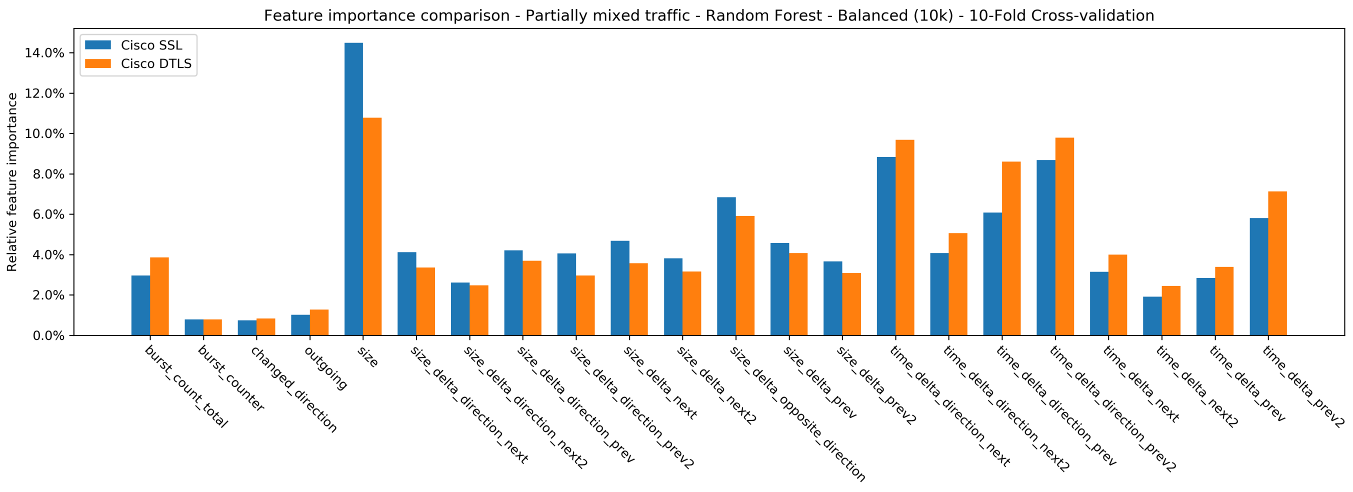


Figure B.3: Comparison of the feature importance for partially mixed traffic

B.3. Entropy based buckets

B.3.1. Size buckets

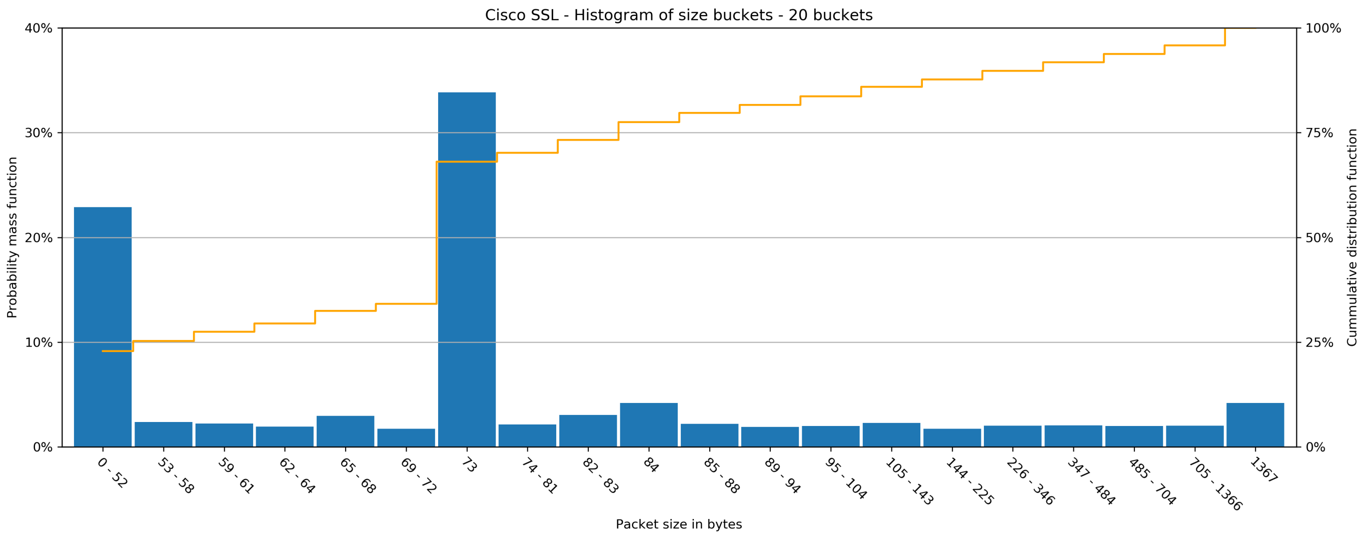


Figure B.4: Entropy size bucket distribution for Cisco SSL VPN

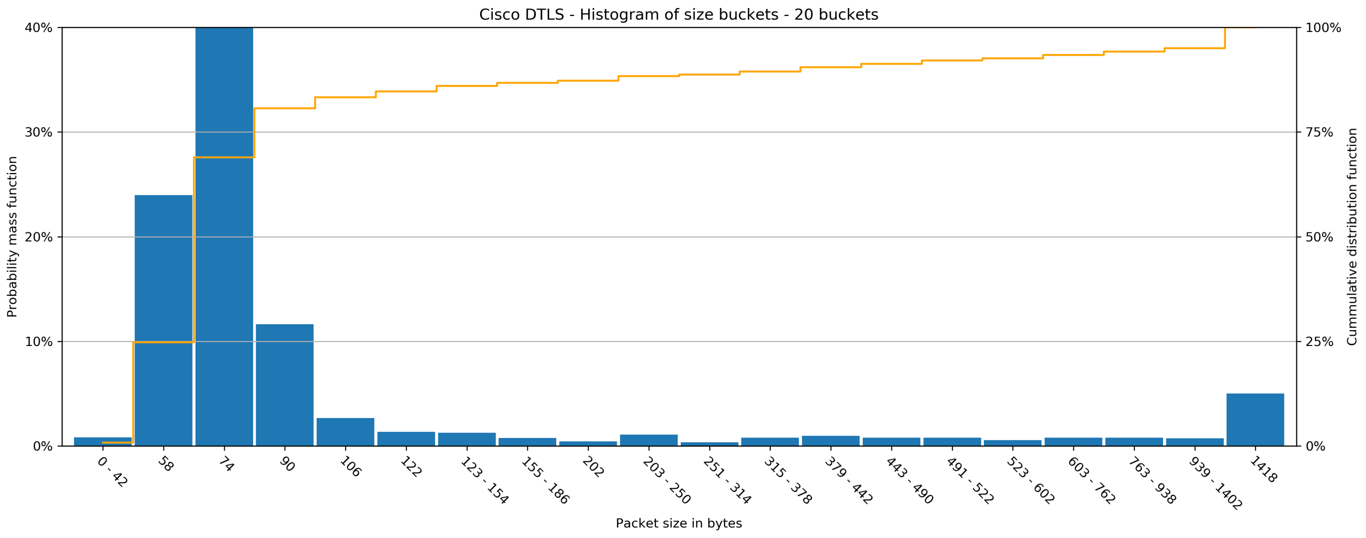


Figure B.5: Entropy size bucket distribution for Cisco DTLS VPN

B.3.2. Time buckets

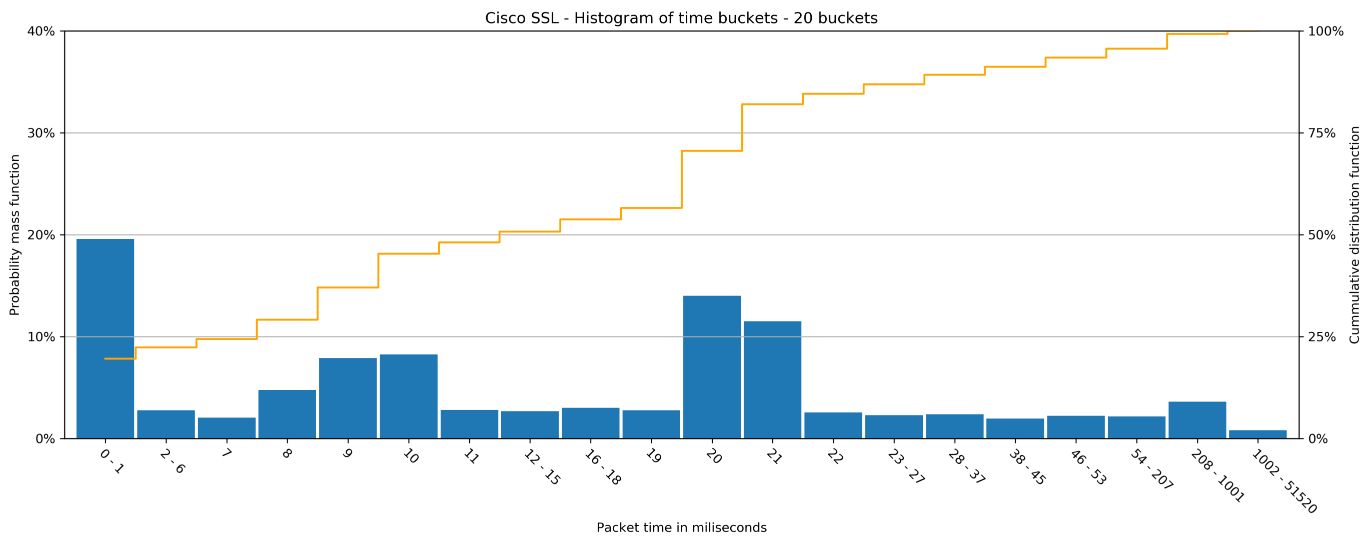


Figure B.6: Entropy time bucket distribution for Cisco SSL VPN

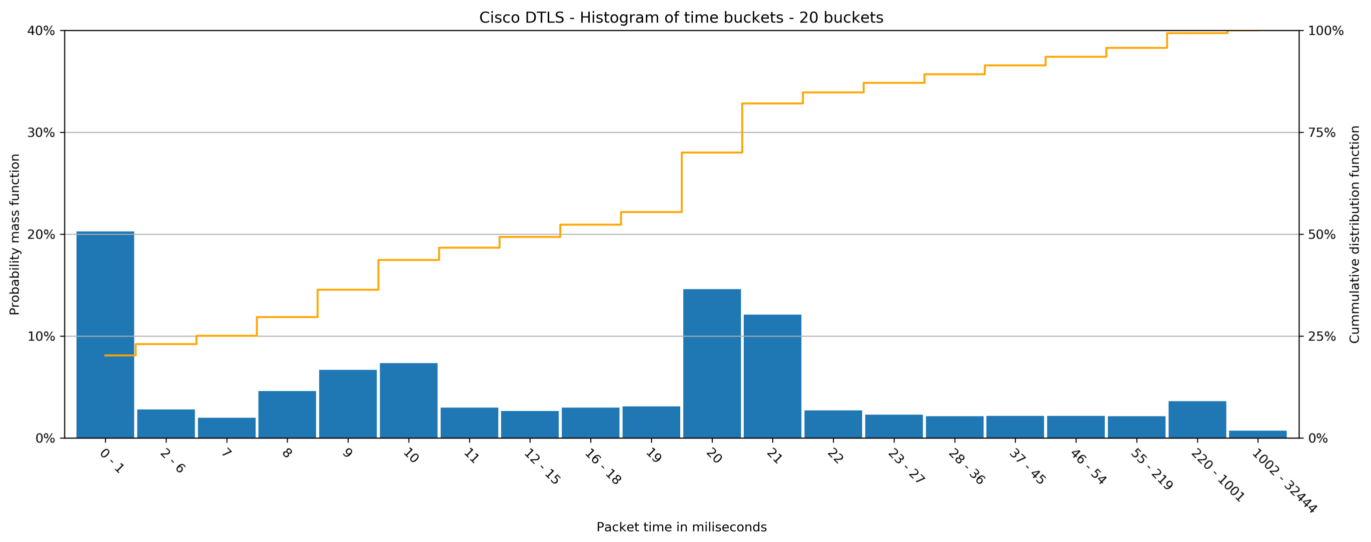


Figure B.7: Entropy time bucket distribution for Cisco DTLS VPN

C
List of features

The final set of features (without Markov scores) consist of:

Feature name	expected value range	explanation
size	28 - 1472	Integer - Original size in bytes(decapsulated)
outgoing	0 - 1	Boolean - Direction, true if outgoing packet
time_delta_prev	0.000001 - 10	Decimal - Delta time to previous packet
time_delta_next	0.000001 - 10	Decimal - Delta time to next packet
time_delta_direction_prev	0.000001 - 10	Decimal - Delta time to previous packet in same direction
time_delta_direction_next	0.000001 - 10	Decimal - Delta time to previous packet in same direction
size_delta_prev	0 - 1444	Integer - Delta size to previous packet
size_delta_next	0 - 1444	Integer - Delta size to next packet
size_delta_direction_prev	0 - 1444	Integer - Delta size to previous packet in same direction
size_delta_direction_next	0 - 1444	Integer - Delta size to previous packet in same direction
time_delta_prev2	0.000001 - 10	Decimal - Delta time to 2 packets back
time_delta_direction_prev2	0.000001 - 10	Decimal - Delta time to 2 packets back that were in the same direction
time_delta_next2	0.000001 - 10	Decimal - Delta time to 2 packets in front
time_delta_direction_next2	0.000001 - 10	Decimal - Delta time to 2 packets in front that are in the same direction
size_delta_prev2	0 - 1444	Integer - Delta size to 2 packets back
size_delta_next2	0 - 1444	Integer - Delta time to 2 packets in front
size_delta_direction_prev2	0 - 1444	Integer - Delta size to 2 packets back that were in the same direction
size_delta_direction_next2	0 - 1444	Integer - Delta size to 2 packets in front that are in the same direction
burst_count_total	1 - 50	Integer - Total number of uninterrupted packets in the same direction
size_delta_opposite_direction	0 - 1444	Integer - Delta size to previous packet in opposite direction
changed_direction	0 - 1	Boolean - Check if direction has changed since previous packet
burst_counter	1 - 50	Integer - Counter for number of consecutive packets in the same direction

Table C.1: Overview of used features

Bibliography

[1] Aditya Sundararajan, Aniket Chavan, Danish Saleem, and Arif Sarwat. A survey of protocol-level challenges and solutions for distributed energy resource cyber-physical security. *Energies*, 11:2360, 09 2018. doi: 10.3390/en11092360

[2] Yi Pang, Shuyuan Jin, Shicong Li, Jilei Li, and Hao Ren. Openvpn traffic identification using traffic fingerprints and statistical characteristics. In Yuyu Yuan, Xu Wu, and Yueming Lu, editors, *Trustworthy Computing and Services*, pages 443–449, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg

[3] John Althouse, Jeff Atkinson, and Josh Atkins. Open sourcing ja3: Ssl/tls client fingerprinting for malware detection, Jul 2017. URL https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41

[4] k-fold cross-validation. URL https://scikit-learn.org/stable/_images/grid_search_cross_validation.png

[5] Percentage of global population using the internet., 2019. URL https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

[6] Cisco annual internet report - cisco annual internet report (2018–2023) white paper, Mar 2020. URL https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[7] Eric Rescorla. HTTP Over TLS. RFC 2818, May 2000. URL https://rfc-editor.org/rfc/rfc2818.txt

[8] Let's encrypt stats. URL https://letsencrypt.org/stats/

[9] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1323–1338, Vancouver, BC, August 2017. USENIX Association. ISBN 978-1-931971-40-9. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/felt

[10] The national security agency: Missions, authorities, oversight and partnerships, aug 2013. URL https://www.nsa.gov/news-features/press-room/Article/1618729/

[11] John L. Gustafson. *Moore's Law*, pages 1177–1184. Springer US, Boston, MA, 2011. ISBN 978-0-387-09766-4. doi: 10.1007/978-0-387-09766-4_81. URL https://doi.org/10.1007/978-0-387-09766-4_81

[12] pc matic 2020 vpn report - a decade's worth of growth. URL https://www.pcmatic.com/news/vpn_report/

[13] Andrew Hintz. *Fingerprinting Websites Using Traffic Analysis*, pages 171–178. Springer Berlin Heidelberg, 2003. ISBN 0302-9743. doi: 10.1007/3-540-36467-6_13. URL https://dx.doi.org/10.1007/3-540-36467-6_13https://link.springer.com/content/pdf/10.1007%2F3-540-36467-6_13.pdf

[14] Maarten van Dantzig. Identifying cobalt strike team servers in the wild, Feb 2019. URL https://blog.fox-it.com/2019/02/26/identifying-cobalt-strike-team-servers-in-the-wild/

[15] W.K. Meijer. Android app tracking. 2019. URL `https://repository.tudelft.nl/islandora/object/uuid%3Abda285c9-117d-49a3-93a1-2530a07f6cff?collection=education`

[16] 2020 world press freedom index: ”entering a decisive decade for journalism, exacerbated by coronavirus”, Apr 2020. URL `https://rsf.org/en/2020-world-press-freedom-index-entering-decisive-decade-journalism-exacerbated-coronavirus`

[17] Alessandro Margara and Tilmann Rabl. *Definition of Data Streams*, pages 648–652. Springer International Publishing, Cham, 2019. ISBN 978-3-319-77525-8. doi: 10.1007/978-3-319-77525-8_188. URL `https://doi.org/10.1007/978-3-319-77525-8_188`

[18] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy vulnerabilities in encrypted http streams. *Privacy Enhancing Technologies*, 3856:1–11, 2006. ISSN 0302-9743. URL `<GotoISI>://WOS:000239416100001`

[19] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 7(100):2745–2769, 2006. URL `http://jmlr.org/papers/v7/wright06a.html`

[20] Kun Zhou, Wenyong Wang, Chenhuang wu, and Teng Hu. Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks. *ETRI Journal*, 42, 01 2020. doi: 10.4218/etrij.2019-0190

[21] Lulu Guo, Qianqiong Wu, Shengli Liu, Ming Duan, Huijie Li, and Jianwen Sun. Deep learning-based real-time vpn encrypted traffic identification methods. *Journal of Real-Time Image Processing*, 17(1):103–114, 2020. ISSN 1861-8200. doi: 10.1007/s11554-019-00930-6

[22] Sami Zhioua. The web browser factor in traffic analysis attacks. *Security and Communication Networks*, 8(18):4227–4241, 2015. ISSN 1939-0114. doi: https://doi.org/10.1002/sec.1338. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1338`

[23] Jonathan Muehlstein, Yehonatan Zion, Maor Bahumi, Itay Kirshenboim, R. Dubin, Amit Dvir, and Ofir Pele. Analyzing https encrypted traffic to identify user’s operating system, browser and application. pages 1–6, 01 2017. doi: 10.1109/CCNC.2017.8013420

[24] Vafa D. Izadinia, D.G. Kourie, and J.H.P. Eloff. Uncovering identities: A study into vpn tunnel fingerprinting. *Computers and Security*, 25(2):97–105, 2006. ISSN 0167-4048. doi: 10.1016/j.cose.2005.12.008. URL `https://dx.doi.org/10.1016/j.cose.2005.12.008`

[25] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. ACM. doi: 10.1145/2660267.2660362. URL `https://dx.doi.org/10.1145/2660267.2660362`