UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

VPN Traffic Fingerprinting

Lennert Kuypers
Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
Prof. dr. Peter QUAX

2021
2022

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

VPN Traffic Fingerprinting

Lennert Kuypers

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Peter QUAX

Universiteit Hasselt

Masterproef voorgedragen tot het behalen van de graad van master in de informatica

VPN Traffic Fingerprinting

Auteur:

Kuypers Lennert

Promotor:

Prof. dr. Quax Peter

Begeleider(s):

dhr. Di Martino Mariano

Academiejaar 2021-2022

Acknowledgements

I would like to thank my mentor Mariano Di Martino and promotor Prof. dr. Peter Quax for helping me throughout the difficult process of writing this thesis. The weekly meetings were extremely helpful and have resulted in many ideas and insights to use in my thesis, this has been a crucial element in this thesis. Furthermore, the feedback after proofreading my thesis for several times has had a major impact. I am extremely grateful.

Furthermore, I would like to thank my parents and my girlfriend for supporting me in stressful times, which has been very important in order to finish this thesis.

Finally, I would like to thank my friends for occasionally providing some distraction resulting in more energy and drive to proceed with my thesis.

Abstract

Virtual Private Networks (VPNs) are a popular service for protecting the privacy of Internet users. Because of the encrypted communication a VPN provides, it is hard to deduce information about endpoints that are using a VPN. The Onion Router (Tor) is another example of such a privacy-enhancing service. In this thesis, we analyse methods that predict classification types based on an encrypted traffic stream, such as a VPN or Tor traffic stream. Furthermore, we construct our own approach to predict the operating system, web browser and traffic type based on a VPN traffic stream without using machine learning (ML) techniques. Therefore, we analyse traces and apply a feature discovery process with multiple experiments to determine functional features for classification. We then implement a live classifying tool that applies our constructed approach. Finally, we discuss possible mitigations to counter the approach. We also follow another approach by adopting two different state-of-the-art ML classifiers to predict the same classification types, and we compare the performances with our own approach.

Contents

Chapter 1

Introduction

The internet is nowadays an indispensable part of our lives. Lots of essential sectors rely on the internet in some way, going from hospitals and the military to schools and businesses [16]. Individuals typically require an Internet connection to complete common daily objectives such as communicating with friends, attending a meeting for work, or ordering food. According to Broadband Search [11], in 2021, approximately 4.93 billion people, representing 63.2% of the world population, had access to the internet and used it frequently. Furthermore, 90.3% of North America and 87.2% of Europe had Internet access and also used it on a daily basis. Although the internet has numerous benefits, it unfortunately also introduces some concerns. One primary concern involves the online privacy and security of the user. We will now elaborate on privacy and security issues for users concerning the traffic they generate while browsing the internet.

A public hotspot is a severe threat to privacy leakage. [47]. Businesses typically create public hotspots from wireless access points (APs) to offer their customers a simple way to connect to the internet. It usually does not require any form of authentication, so anyone can connect to it. Actually, even if authentication is required, it might still be possible for an adversary to decrypt the encrypted traffic exchanged by users and the AP [41]. As a result, these hotspots are dealing with serious security concerns. Attackers can connect to a hotspot and execute a man in the middle (MITM) attack on users that are connected to the same hotspot. MITM will be discussed in greater detail in Section 2.5. The MITM attack allows attackers to intercept and inspect the internet traffic generated by the victim, which is clearly a violation of the victim's privacy. While connected to a private network, for instance, a typical home network, one is relatively safe from becoming a victim of a MITM attack because credentials are required to enter a private network, implying that an attacker needs to know these before the attack can be executed. Nevertheless, other online privacy issues still arise when connected to a private network. A MITM needs to perform some actions to put itself in the user's path. On the contrary, some entities are already in the path, and they must be in that position to ensure a working connection between the user and some other endpoint. An ISP, for example, is such an entity because a user sends all his traffic to his ISP, which in turn makes sure the traffic is forwarded correctly. Cafes, restaurants, or hotels that provide internet access for their customers are also in the path of their internet users. Although the presence of these entities in the user's path is compulsory, they are perfectly able to inspect the data inside the user's traffic which is yet another privacy concern for the users.

It is clear that the user's privacy needs to be protected from eavesdroppers. The HTTPS protocol is the most widely used solution to this problem. HTTPS was introduced to provide confidentiality and integrity to HTTP traffic by applying cryptography. It provides a means for data to traverse the internet safely in a connection between two endpoints. This way, sensitive informa-

tion that is being exchanged, such as credentials, is safe from eavesdroppers. However, HTTPS does not fully provide online privacy protection. While using only HTTPS, an eavesdropper can still deduce which endpoints the user is visiting by inspecting DNS traffic and/or the destination IP address of a TLS connection. When a user is communicating with a Facebook server over a HTTPS connection, for example, an eavesdropper can still determine that the user is connected to that Facebook server, implying that he is currently using Facebook. This provides enough information for the eavesdropper to analyse the user's browsing behaviour, which is, of course, a privacy issue. To mitigate this, an additional layer of protection, possibly on top of HTTPS, is necessary. One popular technology that can provide this is a Virtual Private Network (VPN), which is discussed extensively in Section 2.3. VPN users redirect all of their traffic through a VPN connection to a VPN server. With this technique, users can also hide the destination IP address from eavesdroppers. Additionally, the user's IP address remains hidden from the other endpoint. This way, a VPN significantly hardens the user's privacy and thus offers a solution to the earlier discussed privacy-related problems that internet users may encounter.
Another popular alternative to VPN is the Tor (The Union Router) network. In essence, the Tor network is a decentralized layer of independent nodes. When a Tor user sends a network packet to some endpoint, it is first routed through a randomly selected set of Tor nodes before being sent to the endpoint by the last Tor node in the set. Additionally, multiple layers of encryption are used and each node only knows the previous and next node in the path. There is no best-of-breed option between VPNs and Tor, each has their advantages and disadvantages [58].

This thesis aims to examine existing and discover new methods that result in privacy leakages of a VPN. We will focus on fingerprinting techniques that reveal information about the user, specifically the operating system, web browser, and traffic type of the user. Moreover, we will explore the effectiveness of applying fingerprinting techniques without the help of machine learning (ML), as ML is used very frequently in the state of the art methods that will be described in Section 3.2. This leads to two different approaches we will follow. We refer to the first approach as the 'manual approach'. In this approach, we try to deduce features for classification ourselves by analysing traces. After that, we classify traces based on the features that we discovered. We refer to the second approach as the 'machine learning approach'. Contrarily to the manual approach, we do not try to discover features in the machine learning approach. Instead, we collect traces and train a ML classifier with these traces. This leads to the definition of the following research questions:

- Is it possible to predict device characteristics of a VPN user based on the traffic that is being exchanged with the VPN server?
  - Is this possible by using a manual approach?
  - Is this possible by using a machine learning approach?
  - How does a manual approach perform compared to a machine learning approach?
- What can VPN users do to protect themselves against the techniques applied in the prediction process?

Chapter 2

Background

This chapter introduces some key aspects that are used throughout this thesis. First, the concepts of cryptography and tunneling are explained, which are essential aspects in the context of a VPN. VPNs will be discussed next and are also the main subject of this thesis. The notion of a man-in-the-middle attack is also crucial because this technique is sometimes necessary to intercept the traffic exchanged inside a VPN connection.

2.1 Cryptography

Cryptography refers to techniques for secure communication and information exchange between two entities. The goal of cryptography is for two endpoints to exchange information safely without in-between third parties being able to compromise that information. Cryptography is said to enable Integrity and Confidentiality: two fundamental properties of the CIA triad [23]. Integrity ensures that unauthorised parties can not modify the exchanged information without the two communicating parties being aware of it. Confidentiality ensures the prevention of disclosure of information to unauthorised parties. Cryptography achieves these properties by applying encryption and decryption. Encryption is the process of encoding information in which the original representation, known as *plaintext*, is transformed to the encoded representation, called the *ciphertext*. Decryption is the opposite of encryption; it converts the ciphertext back to plaintext.

There are two main types of cryptography. The first one is symmetric-key cryptography. This method uses the same key to encrypt and decrypt information. This means that both the sender and receiver uses only one key. A popular symmetric-key algorithm is the Advanced Encryption Standard (AES) [20]. The second type is asymmetric-key cryptography, also known as public-key cryptography. The sender and receiver each have a unique pair of keys: a public key and a private key. Only a private key can decrypt the information that was encrypted with the corresponding public key and vice versa. If an entity wants to send an encrypted message to a recipient, it needs the recipient's public key to encrypt the message. This way, the recipient can decrypt the message successfully with its private key. Note that a private key should never be exchanged with other parties, i.e. it should remain private. A widely used system for public-key cryptography is Rivest-Shamir-Adleman (RSA). [61].

2.2 Tunneling

Tunneling is the process of moving data across a network that separates two networks that want to communicate. It is also often referred to as creating a 'tunnel' across a network. A tunnel is created by applying encapsulation. An encapsulated packet is a packet within another packet;

this means that the header and payload of the encapsulated packet are contained within the payload section of the outer packet. Tunneling has multiple useful applications. One includes creating secure network connections by encrypting the packet that contains sensitive information and encapsulating it. VPNs use this method to interconnect private networks across a public or shared network (often the internet) over a secure tunnel. VPNs will be discussed in greater detail in Section 2.3. Tunneling can also be helpful when the network does not support specific protocols. IP in IP tunneling [3], for example, allows two networks using IPv6 to communicate across an IPv4 network by encapsulating the IPv6 packet in an IPv4 packet. Finally, bypassing firewalls is sometimes possible by using tunneling. In Section 2.4, we will show that the ability to bypass firewalls using a tunnel can be of importance.

2.3 Virtual Private Network (VPN)

A VPN extends a private network across a shared or public network. It enables devices to communicate as if they were connected to the same private network. This is achieved by tunneling the traffic of the VPN user through the public network to its destination, which is the desired private network. Note that to reach this private network, the only possibility is to traverse the shared or public network. When a user connects to a VPN, a secure tunnel is created between the user and the VPN server to exchange traffic over the public internet. Cryptography is used to encrypt and decrypt the traffic sent over the tunnel.

2.3.1 Use cases

From the definition, it is clear that the original use case of a VPN is to connect remotely to an internal private network. Today this is still an important reason for using a VPN, especially in a corporate environment. In the past, being physically present at the office was typically required for employees to perform their jobs. The internet has made it possible and even fairly easy to work remotely, which is generally referred to as 'teleworking'. Teleworking is very different from the traditional approach and has its advantages and disadvantages. It is generally more productive and flexible [8], but unfortunately, it can introduce more stress and isolation [9]. Regardless whether or not teleworking is superior, it became mandatory for a significant amount of staff at the start of the COVID-19 pandemic. In nearly a few days, millions of employees had no choice but to shift to teleworking. This obviously resulted in the increased popularity of teleworking, clearly with the COVID-19 pandemic acting as the main cause [50]. During teleworking, a commonly occurring problem is that certain resources may be only available at the physical work site. For example, an employee might need some files that are available only on a file server in the internal office network. VPN technology provides a way to access remote firewall-protected resources securely and is typically used to solve this problem. This indicates that the gain in popularity for teleworking is expected to cause a gain in VPN usage too. This was demonstrated by [24]. They studied the evolution of VPN traffic rates in Europe during the first year of the COVID-19 pandemic. Figure 2.1 shows the observed changes in VPN traffic rates across different months in the year 2020. It is clear that, especially during working hours, VPN traffic rates have increased since the start of the pandemic. They argue that the main causal factor is the higher demand for VPN solutions to access firewall-protected resources hosted in internal company networks.

Nowadays, VPNs are actually being used for many other purposes as well. [64] conducted a study to get an overview of VPN consumer usage. Among other things, they asked the participants to specify why they used a VPN. The main reasons for VPN usage, as a result of the study, are shown in Figure 2.2. Note that the participants were allowed to choose more than one option.

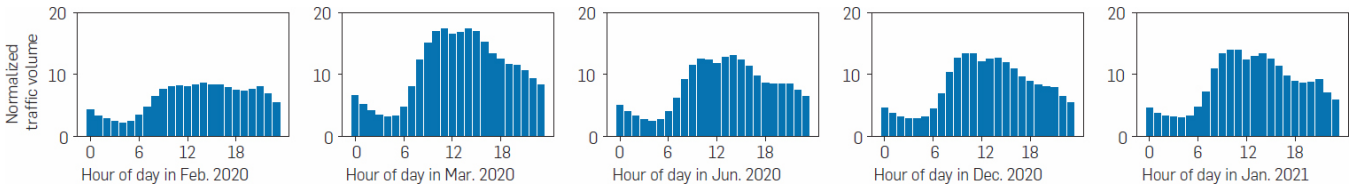


Figure 2.1: VPN traffic evolution during the COVID-19 pandemic [24]

We can observe that VPNs nowadays are used a lot for privacy reasons, including anonymity. This is an interesting finding because VPNs were not originally intended to provide anonymity and/or privacy. Furthermore, a common misconception nowadays is that the word "private" in the VPN initialism refers to privacy, rather than to an internal private network [55]. Although privacy is one of the most popular reasons to use a VPN, the privacy of the user is not always safe from attackers, as we will discuss further in this thesis. A commercial VPN actually introduces an additional privacy concern. All of the user's traffic is forwarded to a VPN server node, controlled by the VPN provider. An important purpose of a VPN is hiding traffic for entities (e.g. an ISP) in the user's path. On the other hand, however, using a VPN actually exposes the plain traffic to the VPN server node, which also represents a central organisation, i.e. the VPN provider, in case of commercial use. This means that users have to trust the VPN provider not to tamper with or log any of their traffic. Many popular providers tackle this issue by guaranteeing a secure infrastructure and no-logs policy. However, in [38] the authors find that this is not always the reality. They analysed 62 commercial VPN services and found that the VPN ecosystem is highly opaque, partly by the lack of practical tools or independent research that audits these claims.
As opposed to a VPN, the Tor network is decentralised, meaning no central organisation is involved. Only the exit node of the Tor network can observe the plain text traffic originally sent by the user, but this node in fact does not know who this traffic belongs to, hence maintaining the user's privacy. Therefore, one could argue that Tor might provide an additional layer of privacy and provides a solution to the central organisation concern of commerical VPNs.
Besides using a VPN for privacy, we can see that it is also popular for security in context of public Wi-Fi as discussed in Section 1, online shopping, and accessing geographically restricted content, i.e. bypassing geo-blocking.

2.3.2 Protocols

VPN protocols are necessary for a VPN to create and manage the secure tunnel for exchanging information safely between two endpoints. In other words, a VPN protocol is responsible for ensuring a secure tunnel between the user and the VPN server. A lot of different VPN protocols exist. We only discuss the most common ones. One of the first available and widely used protocols is the Point to Point Tunneling Protocol (PPTP) [78]. Plenty of security vulnerabilities have been discovered, and the used encryption protocols have become easy to break [33, 63]. This means that PPTP should not be used for security or privacy. It is, however, easy to use, highly compatible, and fast. Therefore it is still in use these days by entities that are less concerned with privacy and security. Besides, PPTP is also still in use in legacy systems that have not been changed. [37]. Another protocol is Internet Key Exchange Version 2 (IKEv2) [22]. IKEv2 establishes shared security attributes (i.e. keys in this case) between remote entities so these attributes can be used later for secure communication; this is called Security Association. IKEv2 typically uses Internet Protocol Security (IPSec) to tunnel data between devices securely. IPSec is a protocol suite that provides secure encrypted communication at layer 3. It needs to be implemented partly in the kernel, so to be able to use IPSec, one needs to verify that the kernel supports it. The Layer 2 Tunneling Protocol (L2TP) [72] creates a layer 2 tunnel between an L2TP Access Concentrator (LAC) and an L2TP Network Server (LNS). The tunnel encap-

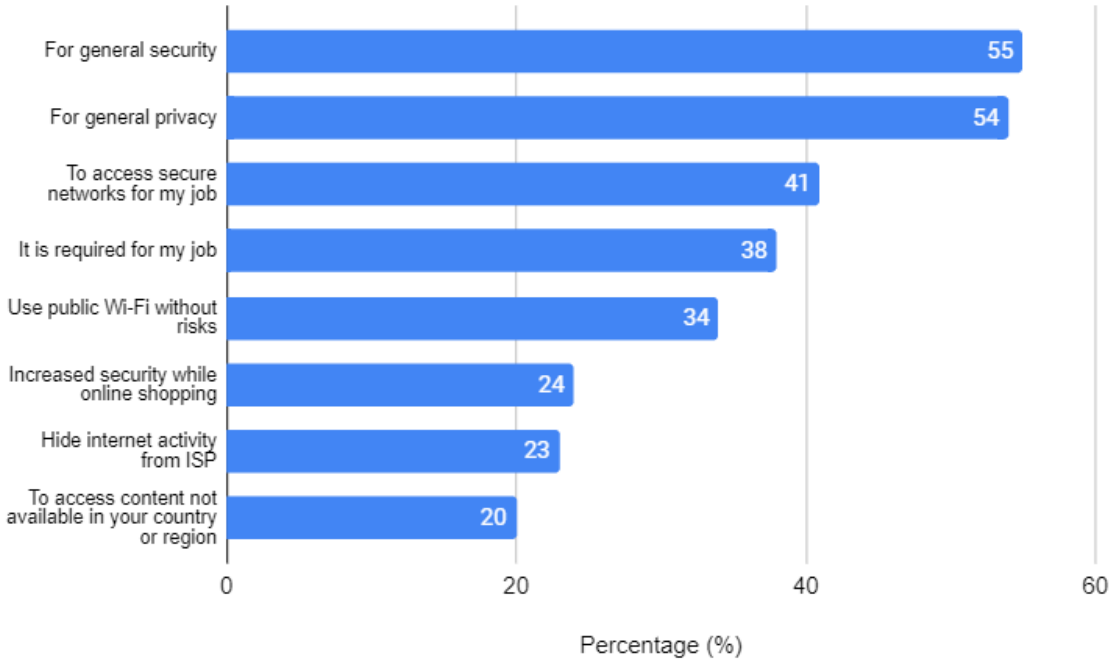


Figure 2.2: The main reasons for using a VPN [64]

sulates L2TP frames in UDP packets. Note that L2TP does not provide any kind of security attributes, so in order to make the tunnel secure, one needs to use L2TP in combination with other protocols, typically IPSec. IKEv1 or IKEv2 is often used for Security Assocation. The Secure Socket Tunneling Protocol (SSTP) uses a TLS channel over port 443. The advantage of this method is that the tunnel traffic can pass through basically all firewalls because port 443 (HTTPS) is very rarely blocked.

Finally, there is the OpenVPN protocol [52] which is the focus of this thesis. We will discuss this protocol in great detail in Section 2.4.

2.4 OpenVPN protocol

OpenVPN [52] is a very popular open-source VPN protocol with over 50 million downloads [52]. In the security context, its open-source characteristics allow people (especially security experts) across the globe to contribute to keeping it as secure as possible. OpenVPN uses the OpenSSL library [1] and TLS protocol to apply cryptography. It fully runs in user space and uses a virtual network adapter as an interface between the OpenVPN software and the operating system. As a result, any operating system that is compatible with a virtual network adapter can run OpenVPN [17]. Additionally, the installation of OpenVPN client software is required to use the OpenVPN protocol.

2.4.1 Virtual network adapters

The traffic flow of OpenVPN data within the VPN client machine is visualised in Figure 2.3. First, an application triggers the kernel to send a packet. The kernel adds the network headers in the networking stack and passes the constructed network packet to the virtual network

[1] https://www.openssl.org/ (accessed on 20-05-2022)

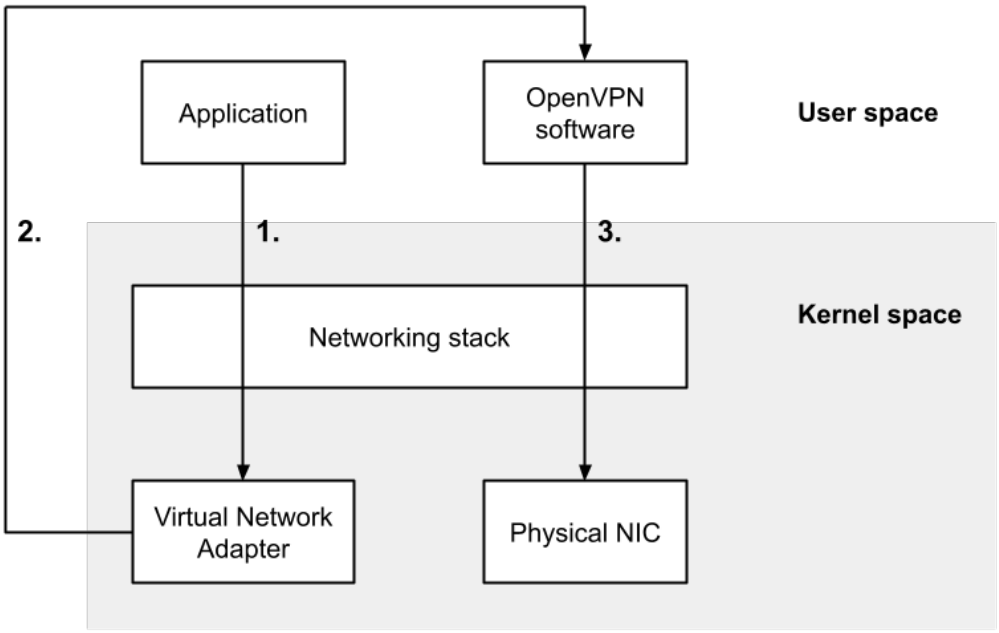


Figure 2.3: Local traffic flow via OpenVPN

adapter. A virtual adapter is essentially a software application, managed by the kernel, simulating a physical network adapter; it is often used to pass received network packets back to specific software applications (in this case the OpenVPN software) instead of sending it over an outgoing link. Generally there are two types of virtual network interfaces. The first one is the TUN interface, which operates at the IP level (layer 3). A TUN interface only accepts IP packets. The second one is the TAP interface, which operates at the Ethernet level (layer 2). Both can be used for specific use cases of a VPN. If one needs to connect two remote LANs using a VPN connection, a TAP interface needs to be used because layer 2 traffic, such as the Address Resolution Protocol (ARP), must be passed across the LANs. On the other hand, if a VPN is used to browse the internet privately, the TUN interface is the better alternative because it does not pass unnecessary layer 2 traffic over the VPN connection resulting in better efficiency.

In the specific case of OpenVPN, the virtual network adapter passes the received network packet to the OpenVPN software, i.e. back to user space. The OpenVPN software then encrypts the entire packet and triggers the kernel to send the result as payload to the IP address of the VPN server, hence passing the payload through the networking stack to the physical Network Interface Card (NIC). Finally, the received packet is put onto the outgoing data link and thus leaves the host machine.

2.4.2 Data and control channel

OpenVPN uses two communication channels during a session: the control channel and the data channel. The control channel utilises a TLS connection to handle authentication, key negotiation, and configuration between two OpenVPN endpoints. TLS Crypt, which applies symmetric encryption using pre-shared keys, is used on top of the TLS connection for an extra layer of security as protection against TLS-level attacks. The data channel handles the secure tunnel in which the actual network packets are transported. Furthermore, the control channel provides

the Security Association for the data channel to encrypt the packets. OpenVPN supports a wide range of symmetric encryption ciphers and hashing algorithms available from the OpenSSL library. The purpose of the encryption ciphers is to encrypt some payload while a HMAC function utilises the hashing algorithms to authenticate packets. The data channel encryption cipher used for encrypting the network packets can be chosen freely from the available ciphers. However, client and server both need to support and allow the chosen cipher. The latest OpenVPN version recommends and defaults to the AES-256-GCM symmetric cipher [62] for the data channel [2] [3].

2.4.3 UDP vs TCP tunnel

An OpenVPN packet is encapsulated in the payload at the transport layer level. Two transport layer protocols are supported for this purpose: TCP and UDP. The Internet Assigned Numbers Authority (IANA) has assigned port 1194 to OpenVPN for both UDP and TCP [34]. This port is considered the default port for UDP, but the TCP port for OpenVPN defaults to 443. Note that OpenVPN can be configured manually to run on any port if one wishes to omit the default ports.
When using OpenVPN, one must select either TCP or UDP for the tunnel. A TCP tunnel is the obvious choice for bypassing firewalls if default ports are used since HTTPS also runs on port 443. Besides, it is relatively easy to block OpenVPN over UDP because it uses a port specifically dedicated to OpenVPN as mentioned earlier. Note that the encryption used by OpenVPN over TCP is not exactly the same as the encryption used by HTTPS, so it is possible to distinguish these two by applying Deep Packet Inspection. Whilst TCP is better for bypassing firewalls, it also has a significant disadvantage: when a TCP tunnel is transporting a TCP connection inside it, one can possibly experience a network slowdown. This is called the TCP Meltdown problem [15]. The network slowdown can occur when both of the stacked TCP connections are retransmitting packets, but the outer connection is slower (higher timeouts) than the inner connection. This causes the inner connection to queue up more retransmissions than the outer layer can handle, resulting in a significant latency increase. A performance comparison between TCP and UDP tunnel connections was conducted in [14]. They found that a UDP tunnel utilises the link more efficiently and yields improved transfer times and speed compared to a TCP tunnel. They also confirmed that a TCP connection in a UDP tunnel provides better latency than in a TCP tunnel. The unreliable nature of UDP is clearly beneficial for OpenVPN in terms of performance. Furthermore, using a UDP tunnel for OpenVPN is not problematic for ensuring reliability: if reliability is required, the tunneled TCP connection will take care of it.
We conclude that, considering performance, UDP seems to be the better option. It is, however, relatively easy for firewalls to block a UDP tunnel. That is why most entities, including the OpenVPN team itself [51], advise first to try UDP and to switch to TCP if any firewall issues arise. Furthermore, the connection profiles downloaded from the official OpenVPN Access Server [4] are by default configured to first try UDP.

2.4.4 Encapsulation

When a regular network packet is passed to the OpenVPN software, the software encrypts it with a symmetric-key cipher, as discussed in Section 2.4.2. Note that, when we use the term 'regular packet', we refer to a network packet that has not (yet) been encapsulated or encrypted by a privacy-enhancing technology like a VPN or Tor. After the software has encrypted the

[2] https://community.openvpn.net/openvpn/wiki/CipherNegotiation (accessed on 20-05-2022)
[3] https://openvpn.net/vpn-server-resources/change-encryption-cipher-in-access-server/ (accessed on 20-05-2022)
[4] https://openvpn.net/access-server/ (accessed on 20-05-2022)

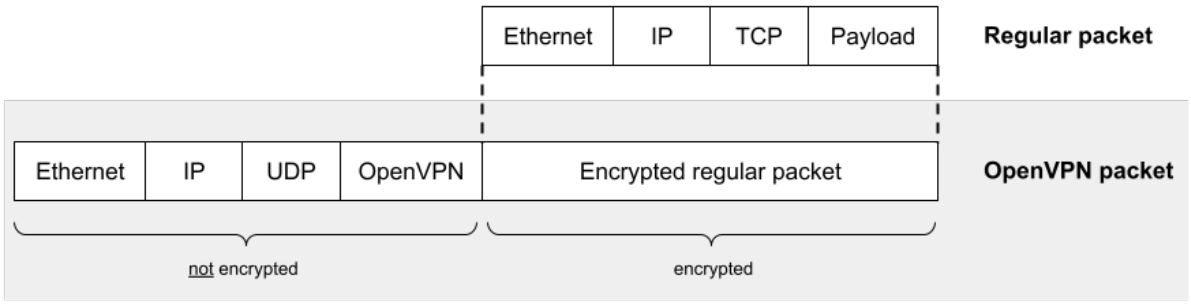


Figure 2.4: OpenVPN (UDP tunnel) encapsulation of regular TCP packet

regular packet, the encrypted packet is encapsulated by adding the necessary headers to create a new packet, i.e. the encrypted packet is used as payload in a new, encapsulating packet. In detail, Ethernet, IP, UDP, and OpenVPN headers are added to create the final OpenVPN packet, assuming that OpenVPN is using an UDP tunnel. Figure 2.4 contains a visualisation of a regular TCP packet and its corresponding encapsulated OpenVPN packet. The final OpenVPN packet is a representation of the packets that are being transmitted by the OpenVPN client machine. Therefore, it also represents the traffic that can be observed by a MITM between the OpenVPN client and server. For clarification, as indicated in Figure 2.4, the headers that were added to create the encapsulation packet are not encrypted, hence visible to a MITM. On the contrary, the encapsulated packet is encrypted, hence not visible to a MITM.

2.5 Man in the middle attack

A man in the middle (MITM) is an entity that has inserted itself in the connection path between at least two endpoints that are communicating. These endpoints are typically not aware of the presence of the MITM, but this is not necessarily the case. An internet service provider (ISP), for example, could also act as a MITM, but it is evident that our Internet traffic passes through our ISP. There are numerous methods available for an attacker to obtain a MITM position; a few possibilities include ARP poisoning and setting up a rogue access point.
A MITM can generally behave either as a passive MITM or an active MITM. The former only eavesdrops on the network traffic of the target connection without altering it in any way. The latter actively manipulates the packet stream by changing, dropping, or adding packets. An active MITM approach is often used to pretend to be one of the endpoints, meaning that the other endpoints are actually communicating with the MITM without knowing it.
A MITM is most destructive when the endpoints communicate via protocols that do not apply cryptography. In that case, data is sent in plain text over the internet, which is generally a terrible idea. For instance, if a passive MITM is eavesdropping an HTTP connection between a client and a server, the MITM can easily obtain sensitive information (e.g. passwords). Likewise, an active MITM can easily pretend to be another entity without the endpoints being aware of it if no cryptography mechanism is used. This clearly indicates that encryption should be applied to data exchanged between endpoints, making it substantially harder for a MITM to obtain valuable/sensitive information or tamper with the communication. For instance, when using a VPN for privacy or security reasons, the data in the secure tunnel is encrypted to prevent MITM entities, such as an ISP, from tracking what the user is doing.

Chapter 3

Fingerprinting intercepted traffic

The internet was not built to be secure; its initial purpose was to connect many people around the globe [56]. Consequently, the most common protocols are not designed with security in mind. The HTTP protocol, for example, sends all data in plain text over the internet even if it contains sensitive information such as credentials, making it extremely easy for an entity in the connection path to gather sensitive information about the user. Cryptography is a well-known method to counter this. If two endpoints use cryptography to encrypt their communication data, it will be more rigid for eavesdroppers to derive sensitive information from the observed encrypted traffic. Nonetheless, there are still techniques available to gather interesting information from an encrypted traffic stream by entities that are not able to remove the encryption layer. A popular technique among these is fingerprinting. Fingerprinting allows an adversary to obtain additional information about the sender of a traffic stream, often encrypted, by applying pattern recognition. Fingerprinting can be performed in lots of different ways. One can, for example, simply look at plain text attributes of the intercepted traffic and, solely based on this, deduce information about the user. On the other hand, there are much more complex methods of fingerprinting, e.g. only considering packet sizes of encrypted traffic and applying ML. The specific information that an adversary wants to obtain by applying fingerprinting techniques is usually in the form of some classification. A frequently encountered classification category is Website Fingerprinting (WF). The main objective of WF is to identify which webpage a user is visiting based on an encrypted traffic stream that the user is generating. Thus, WF aims to classify webpages. More specifically, WF typically aims to either classify multiple webpages of one specific website or classify multiple websites by considering at least one webpage per website. Another popular classification category is traffic categorization. Traffic classification can be categorized based on different purposes. Some options include protocol encapsulation (e.g. VPN vs HTTPS), specific applications (e.g. YouTube, Vimeo), and general application type (e.g. Streaming, Browsing). Additional less frequently occurring categories are operating systems (e.g. Windows, Linux) and web browsers (e.g. Chrome, Firefox). Note that web browser classification is not to be confused with 'browser fingerprinting' [28, 36, 45], which is a technique that websites (as endpoints) use to establish a unique fingerprint of each user's browser in order to track users across sessions.

This chapter focuses on fingerprinting techniques to derive classification-based information about the user deduced from a traffic stream on which some privacy-enhancing technology is applied, such as HTTPS, a VPN, or Tor. The used technology affects the set of fingerprinting techniques necessary to determine some piece of information, as well as their effectiveness. For instance, HTTPS and even a VPN in some scenarios reveal plain text attributes in traffic that can be used for certain classifications, as will be discussed in Section 3.1. To summarize, the selection of fingerprinting techniques, if necessary, to determine some piece of information about the user

(e.g. visited domains, browser type) depends on what privacy-enhancing technology is being used.
In the upcoming sections, we will discuss state-of-the-art (fingerprinting) techniques used to reveal information about the user of some privacy-enhancing technology.

3.1 Plain text attributes

Sometimes, the desired information about a user can be deduced simply by only considering plain text attributes of the intercepted traffic. In this section, we will discuss some examples that use this approach to perform fingerprinting attacks.

HTTPS only encrypts the application layer (HTTP protocol) of HTTP traffic and a client using only HTTPS directly communicates to the destination server. If HTTPS is used, some plain text attributes that can be valuable for classification are available. For instance, the actual domains can be deducted directly from the client's traffic by inspecting DNS packets or the Server Name Indication (SNI) field in the TLS Client Hello, assuming these are sent in plain text. Actually, even if the domain name is not directly deductible because DNS and TLS Client Hello packets are hardened with DNS over HTTPS [32] and Encrypted SNI [59], there still exist fingerprinting methods [18] that can deduce the domain name by considering the plain text destination IP address. Of course, if the domain is leaked in plain text to a MITM, there is no need to use more sophisticated fingerprinting techniques to determine the visited domains. As a result, one can deduce the domain names from the user's traffic by considering plain text elements residing in the traffic and use this for classification of domain names [27, 42]. Moreover, domain names can also possibly be used for other classifications, such as OS classification by looking for domains that correspond to update services for some specific OS.

Several methods have been proposed to classify OSes based on plain text header values. Aksoy et al. [4] targeted packet headers of TCP, IP, and UDP sections. The considered OSes include Raspberry OS, Xubuntu 14.04, Windows 7, Windows 8, Mac OSX Elcapitan and Mac OSX Lion. Genetic algorithm feature subset selection is used to determine relevant packet header features. The remaining features are then used to train the following ML algorithms: J48, RandomForest, OneR, and ZeroR. Anderson et al. [7] focus on TLS, TCP, IP, and HTTP headers and include different versions of Windows OS and OS X. Chen et al. [13] focused on TCP and IP headers and Windows, iOS, and Android OSes. Considering individual features, they found the IP time to live (TTL), TCP timestamp, and TCP window size scale factor to be the most accurate ones.

Althouse et al. created a method for TLS fingerprinting called JA3 [6]. It focuses on plain text fields in TLS negotiation packets, more specifically the TLS Client Hello and TLS Server Hello. JA3 then creates an MD5 hash for both sets of attributes, yielding unique hashes representing the sender (origin of Client Hello) and receiver (origin of Server Hello). From the TLS Client Hello, JA3 specifically collects the TLS version, accepted ciphers, extensions list, elliptic curves, and elliptic curve formats. From the TLS Server Hello, the collected attributes include TLS version, accepted ciphers, and extensions list. JA3 can be used to detect malware applications, because the malware client and command/control servers will always communicate in exactly the same way, hence generating the same unique hashes. Furthermore, the client side hash can also be used to detect web browser client type because different client types typically do not specify the attributes in exactly the same manner; specifying the available ciphers in another order is sufficient to generate a different hash.

Contrarily to HTTPS, VPNs and Tor encrypt all of the user's traffic and redirect it to one or multiple nodes before sending the traffic to the actual destination specified by the user. This clearly provides a higher level of privacy/anonymization and the encryption of basically all of the user's traffic should not expose any valuable plain text information to deduce information about the user. However, especially for VPNs, some VPN clients still leak (plain text) data that can be used for classification, often caused by implementation errors. It was found that VPN implementations sometimes do not manipulate the IPv6 routing table of the client, hence not redirecting IPv6 traffic through the VPN tunnel or blocking IPv6 traffic if the VPN client does not support IPv6; besides, leakage of DNS traffic is also possible. [29, 35, 54, 74]. [39] tested 43 VPN services for IPv6 and DNS leakage, they found that 2 services leaked DNS traffic and a remarkable amount of 12 services leaked all IPv6 traffic. This leakage of this data can be a severe issue, especially if the main goal of the user is obtaining privacy/anonymity. An ISP, for instance, can inspect the leaked IPv6 and/or DNS traffic and determine the domains the user is visiting or create a unique fingerprint with JA3 based on the TLS Client Hello packets (sent over IPv6), as discussed earlier in this section. This clearly violates the desired the privacy/anonymity of the user.

3.2 VPN and Tor traffic

In the previous section, we discussed that in some scenarios it is possible to apply fingerprinting simply by considering plain text attributes. In this section, contrarily, we only consider VPN and Tor traffic streams, thereby assuming that no plain text elements are available in the traffic to help with the classification process. As a result, we are forced to perform classification by analysing the encrypted content of the packet, hence applying more complex fingerprinting techniques. We will focus specifically on techniques that involve the use of ML models, which is commonly used in the state of the art. Assuming that we cannot decrypt the encrypted content as a MITM, there is only a limited set of useful features to work with. Some features that can be derived from an encrypted stream include packet arrival times, packet sizes, direction of the packets (if both directions of communication are considered), and amount of packets. Optionally, one can also choose not to extract any features at all manually and just use the raw traffic trace. We will now discuss the current state of the art of performing classification solely based on the encrypted part of the intercepted traffic stream. Note that most research of this kind mainly focuses on website fingerprinting and traffic type fingerprinting, meaning that there exists little work for other classification types. The discussed research is also summarised in Table 3.1

Gerard-Gil et al. [30] propose a classification method to characterize VPN traffic and possibly other types of encrypted traffic. Only time-related features are used, hence ignoring packet size features. They generated and published the ISCX VPN-nonVPN dataset (ISCXVPN2016) [1], containing 7 categories of internet traffic, captured over a regular session as well as over a OpenVPN (UDP tunnel) session, including Email, Chat, Streaming, File transfer, VoIP, and P2P. The C4.5 Decision Tree and K-Nearest Neighbors (KNN) machine learning algorithms are used on the generated dataset to perform traffic classification. Additionally, a similar method was used to characterize Tor traffic instead of VPN traffic. The Tor-nonTor dataset (ISCXTor2016) [2] was created containing the same traffic types as the VPN dataset. Concerning the choice of ML algorithms, two extra algorithms were considered, namely Zero R and Random Forest. The most important contribution of the authors is the provision of extensive datasets for VPN and

[1] https://www.unb.ca/cic/datasets/vpn.html (accessed on 20-05-2022)
[2] https://www.unb.ca/cic/datasets/tor.html (accessed on 20-05-2022)

Tor traffic, which is crucial to perform classification with ML.

Source	Traffic	Features	Classification	ML algorithm
Gerard-Gil et al. (2016)	VPN	Timing	Application type	C4.5 Decision Tree K-Nearest Neighbors
Gerard-Gil et al. (2016)	Tor	Timing	Application type	C4.5 Decision Tree K-Nearest Neighbors Zero R Random Forest
Shapira et al. (2019)	VPN	Timing Size	Application type	LeNet-5
Rimmer et al. (2018)	Tor	Raw trace	Webpage	Stacked Denoising Autoencoder Convolutional Neural Network Long-Short Term Memory
Bhat et al. (2019)	Tor	Raw trace Timing Direction Cumulative statistical features	Webpage	Var-CNN (ResNet-18 architecture)

Table 3.1: Comparison of current state of the art

Shapira et al. [65, 66] use the ISCX VPN-nonVPN dataset together with an additional small packet capture to perform traffic classification. They apply a remarkable pre-processing method to the data by converting it to a picture; they name the resulting picture a 'FlowPic'. Specifically, the picture is a two dimensional 1500x1500 histogram in which the X-axis and Y-axis are defined by packet arrival time and packet size features respectively. The maximum packet size is chosen to be 1500 bytes, which is the Ethernet MTU value, hence limiting the Y-axis value to 1500. The packet arrival times are normalized by substracting the time of arrival of the first packet in the flow. To obtain a square, for simplicity, the maximum X-axis value is also set to 1500. This requires a second normalization step for the packet arrival times in which the initial range is mapped to the desired [0, 1500] range. The conversion to pictures allows the authors to take advantage of known image classification deep learning techniques, typically including Convolutional Neural Networks (CNNs). More specifically, they choose a LeNet-5 style architecture [46].

Rimmer et al. [60] perform WF on Tor traffic with multiple Deep Neural Networks (DNNs): Stacked Denoising Autoencoder (SDAE), CNN, and Long-Short Term Memory (LSTM). They emphasize the importance of feature engineering, which includes the process of selecting what hyperparameters to use for a certain ML algorithm. To evaluate their approach, they created a dataset, both for the closed and open world scenario, in a very comprehensive manner. For the closed world scenario, dataset contains a total of 900 monitored sites, each with 2,500 traces. In the open world case, approximately half a million additional unmonitored sites were added, each with just 1 trace of the home page visit. All pages were extracted from the Alexa list of most popular sites. Alexa unfortunately ended its service. Whilst feature engineering is typically performed manually, they choose a different approach by automating this process. They first choose a representative subsample of the dataset and a set of possible hyperparameters for the ML algorithm. Then, these are fed to an automated hyperparameter tuning process which selects the best combination of hyperparameters based on performance on the specified subsample. They compared their approach's performance with the state-of-the-art performances at the time. The SDAE had better results for their largest closed world dataset and both the SDAE and CNN performed slightly better in the open world scenario. It is remarkable that the raw traces are used as input for the ML algorithms, hence feature extraction is not performed.

Another WF attack was performed by Bhat et al. [10]. A Var-CNN is created to perform classification. Var-CNN's base architecture uses ResNets [31], which are state-of-the-art convolutional

neural networks (CNNs) for image classification purposes. For this use case, specifically ResNet-18 is chosen, which is the smallest version of ResNet with 18 layers. Hyperparameter tuning is also applied, similar to the process described in the work of Rimmer et al. which we discussed earlier in this section. An interesting choice of the authors is the combining of automated feature extraction of the raw trace with manually deduced timing, direction, and cumulative statistical features such as total amount of incoming & outgoing packets and ratio of incoming to total packets. They show that combining these sets of features yields the best performance. Tests were performed on the dataset created by Rimmer et al. They conclude that their approach achieves over 1% better true positive rate (TPR) and four times lower false positive rate (FPR) than prior state of the art in open-world settings with large amounts of data. Furthermore, they argue that Var-CNNs perform even better in low-data scenarios.

Chapter 4

VPN operating system, browser, and traffic type fingerprinting: manual approach

The state-of-the-art techniques discussed in Chapter 3 all rely on some form of machine learning. This chapter proposes an approach to fingerprint encrypted traffic without utilising any machine learning techniques. This strategy also allows us to examine the feasibility of fingerprinting encrypted traffic without using machine learning. We will focus specifically on encrypted traffic that is sent over a VPN tunnel. The threat model is shown in Figure 4.1. Supporting the threat model, we make the following assumptions to define the situation we are targeting:

- A VPN client is connected to a VPN server and redirects its traffic through the VPN connection to browse the internet using a desktop web browser.
- The VPN client only uses one type of browser at a time.
- the VPN client is not generating a significant amount of irrelevant background traffic.
- An attacker is in a MITM position and is able to intercept the encrypted traffic that the VPN client and VPN server are exchanging.
- The attacker can only observe the intercepted encrypted traffic, i.e. the attacker does not have any additional information to work with.

From Chapter 3, we can see that traffic and webpage classification are popular classification choices for fingerprinting. Unfortunately, relatively few sources have made an effort to classify web browsers and operating systems. Concerning web browsers, however, research has been done to analyse the effect of web browser type on the accuracy of website fingerprinting attacks [5, 76, 77]. These studies show that the choice of web browser type affects the results of the fingerprinting attack. We consider this an indication that differences between web browsers can be observed during fingerprinting attacks, hence making it possible to classify web browsers.

Our goal is to classify the web browser, operating system, and traffic type solely based on the intercepted encrypted traffic. We choose Google Chrome, Microsoft Edge, and Mozilla Firefox as web browser classification items. According to StatCounter Global Stats, Chrome, Edge, and Firefox respectively represented 64.91%, 9.61%, and 9.47% (all together 83.99%) of worldwide desktop browser market share in February 2022 [1].

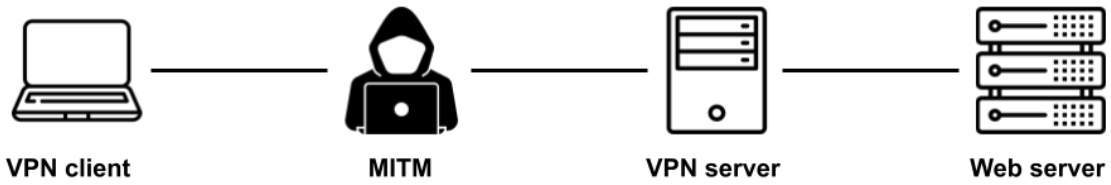


Figure 4.1: Threat model

Web browser	Operating system	Traffic type	VPN
Google Chrome Microsoft Edge Mozilla Firefox	Windows 10 Ubuntu 20.0 LTS	Browsing YouTube (on-demand) Twitch (live)	OpenVPN (UDP)

Table 4.1: Overview of chosen parameters

We choose Windows 10 and Ubuntu 20.04 LTS (Linux) for the operating systems. Note that in the remaining part of this thesis, we will refer to the latter operating system as linux. StatCounter indicates that the Windows and Linux OS represent 74.79% and 2.49% respectively of the worldwide OS market share [2]. Although Linux only represents a relatively small share, it still has the third biggest share; only OS X has a bigger share. However, because we are not in possession of an OS X compatible device, we choose to use Linux instead. Ubuntu 20.04 LTS was specifically chosen because it is currently the latest Ubuntu version and Ubuntu is one of the most popular available Linux distributions.
The traffic types include browsing, YouTube, and Twitch. With the term 'browsing', we include visiting (random) websites and navigating through the webpages by often clicking a random link available on the webpage. Additionally, we will consider streaming in the context of the client receiving a video stream; therefore, our client is not the generator of the stream. Moreover, we will use on-demand streaming for YouTube and live streaming for Twitch. YouTube and Twitch are specifically chosen because YouTube uses an HTTP/3 (together with the QUIC transport protocol) approach for streaming. In contrast, Twitch uses a lower HTTP version approach (TCP transport protocol), yielding an interesting comparison in the difference of generated network packets. We will go into further detail in Section 4.3.1.
Considering the VPN protocol, we will use OpenVPN over UDP as OpenVPN currently is one of the most widely used protocols and UDP is clearly preferred to use for OpenVPN as discussed in Section 2.4. A summary of the chosen parameters is shown in Table 4.1.

4.1 Experiment: visualising browser generated traffic

We start with an experiment in which we generate and analyse traces for all possible combinations of options of different classification types (e.g. Windows-Chrome-Browsing). The experiment aims to get a better understanding of the traces and observe differences between classifications. We use automated web crawlers implemented with the Selenium [1] library to generate traffic traces for all combinations while using an OpenVPN connection. For the web browsers Chrome, Edge, and Firefox, the used versions are 97.0.4692, 97.0.1072.62, and 96.0.1

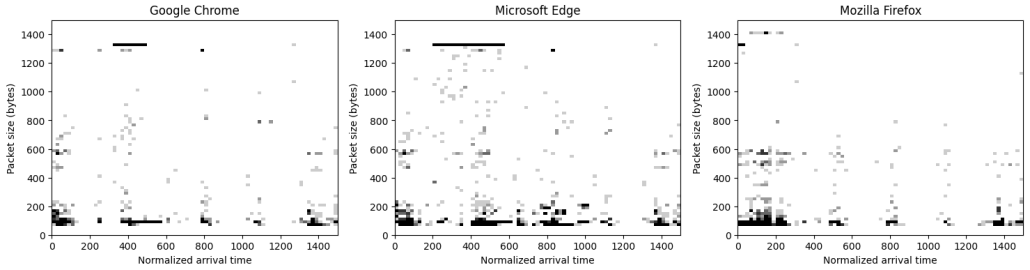
[1] https://www.selenium.dev/ (accessed on 20-05-2022)

Traffic	Instructions
Browsing	1. Start capture 2. Open uhasselt.be 3. Follow some pre-defined links 4. Stop capture
YouTube	1. Open a pre-defined video on youtube.com 2. Wait 5 seconds 3. Start capture 4. Wait 60 seconds 5. Stop capture
Twitch	1. Open a pre-defined livestream on twitch.tv 2. Wait 5 seconds 3. Start capture 4. Wait 60 seconds 5. Stop capture

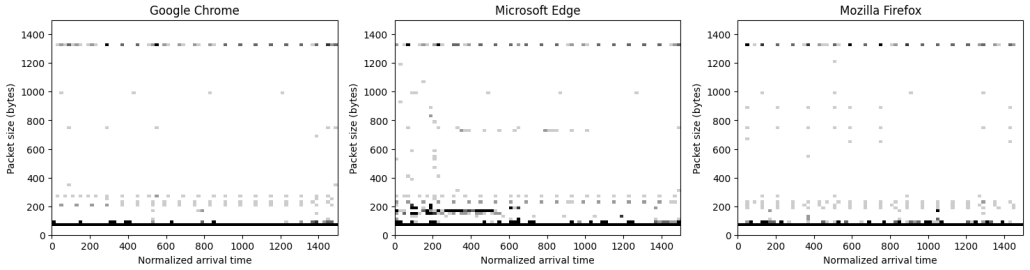
Table 4.2: Instructions used for web crawlers

respectively. The instructions applied by the web crawlers to generate traces for the different types of traffic are shown in Table 4.2. We represent the traces visually by adopting the pre-processing approach 'FlowPic' used by [65] as discussed in Section 3.2. Figure 4.2 shows the produced visualisations.

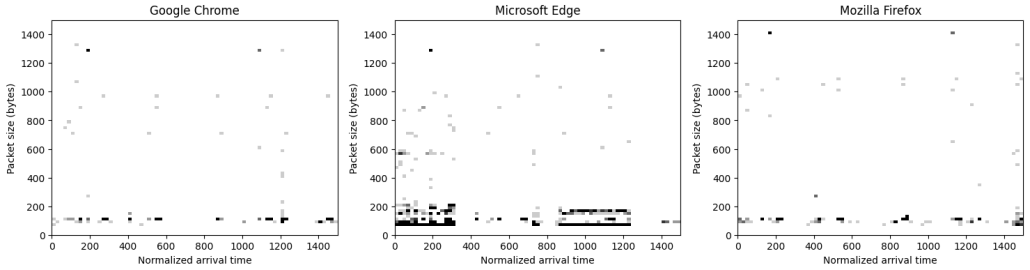
First of all, we see that, generally, the visualisations of corresponding traces across the OSes are very similar, indicating that it is not trivial to distinguish between the OSes based on time and size-related features alone. However, this is still an initial exploration, and we will later argue that there are some features we can deduce from the traces to distinguish between the OSes.
Focusing on the different web browsers, we notice some notable differences. The visualisation of Edge looks denser compared to Chrome and Firefox. This is an observation that we can further analyse by, for example, considering the packet rate of the traces. Additionally, in particular for YouTube and Twitch traffic, Edge contains significantly more packets around the size level of 200 bytes than Chrome and Firefox. Considering the boundaries of the packet sizes, when looking at the browsing and YouTube traffic visualisations in particular, an interesting observation is that the largest encountered packet size for Firefox is clearly more significant than the value for Chrome and Edge. For Twitch traffic, this is not visible. By further analysing the maximum packet sizes of the browsing and YouTube traces, we find that the maximum values for Firefox, Chrome, and Edge are 1451 bytes, 1444 bytes, and 1444 bytes respectively. Note that these are the sizes of the outgoing OpenVPN packets. This is an interesting observation that we will use later on in Section 4.3.
We see pronounced differences between the traces when analysing the different traffic types. The visualisations of browsing traffic show more scattered packet sizes than the ones for YouTube and Twitch; the packet sizes vary over time for browsing traffic while they remain relatively similar over time for YouTube and Twitch traffic.



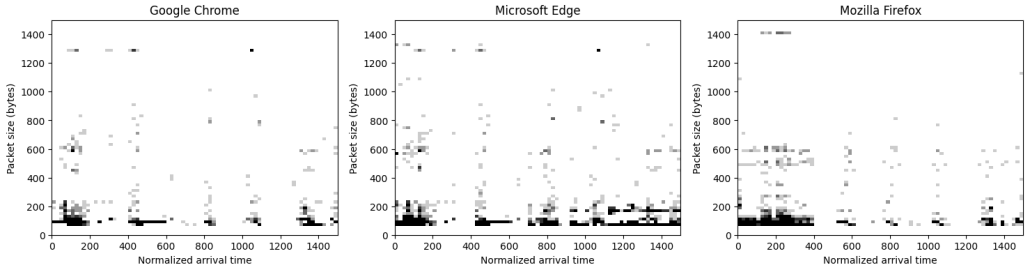
(a) Windows - Browsing



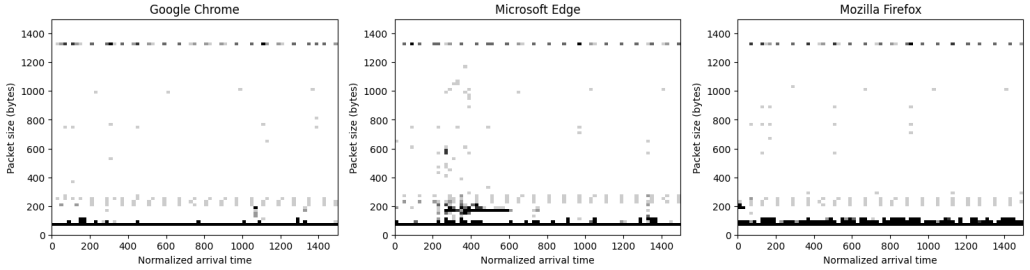
(b) Windows - Twitch



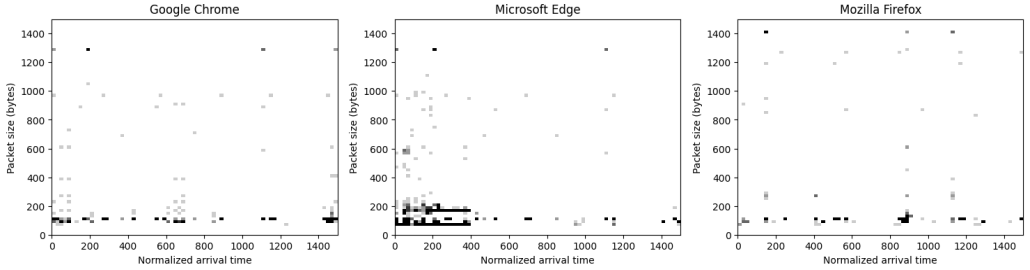
(c) Windows - YouTube



(d) Linux - Browsing



(e) Linux - Twitch



(f) Linux - YouTube

Figure 4.2: FlowPic visualisations of web crawler traces

4.2 Matching OpenVPN packets with regular packets

In this section, we discuss a critical observation that is crucial for the further process of feature discovery. This observation includes that the difference in length, expressed in bytes, between a regular packet and its corresponding OpenVPN packet is a constant value. With this information, we can determine the length of the regular packet that was encapsulated, given its OpenVPN packet. We now elaborate on how this is possible and under what circumstances our claim holds.

In Section 2.4.1, we show how the plain traffic, generated by some application, is passed to OpenVPN software which applies encryption and encapsulation. Recalling Figure 2.4 (in Section 2.4.4), we describe the process of encrypting the regular packet and encapsulating it in greater detail. For the encapsulation part, concretely, an Ethernet, IP, UDP, and OpenVPN header is added to create the OpenVPN packet. These headers have constant lengths of 14 bytes, 20 (IPv4) or 40 (IPv6) bytes, 8 bytes, and 4 bytes respectively. Note that the IP layer can either be IPv4 or IPv6. In the case of IPv6, the header length will always be 40 bytes. When IPv4 is used, the header length is not fixed because multiple options might be included on top of the 20 required bytes of essential information. However, based on our observations, the options are for specific use cases and are often mitigated, resulting in a header length of 20 bytes. Our OpenVPN tunnel uses IPv4 to transport tunnelled packets. We will not consider the possibility that the IPv4 header length might be larger because we rarely encountered occurrences of options usage while performing tests.
From Section 2.4.2, we know that, currently, the default config file of the OpenVPN Access Server specifies using the AES-256-GCM symmetric-key cipher for encryption of the regular packet. This cipher is also used in our OpenVPN setup. An essential characteristic of GCM (Galois Counter Mode) is the fact that it does not apply padding [48], implying that the cipher text has the same length as the plain text. In other words, when encrypting a regular packet, the length will not change. Additionally, the GCM encryption algorithm adds an authentication tag with a fixed length of 6 bytes to the cipher text.

With this knowledge, we can infer that all of the headers, together with the GCM authentication tag, added to the encrypted regular packet (to create the OpenVPN packet) have fixed lengths, all adding up to a total of 52 bytes. Therefore, we can obtain the length of the encrypted regular packet by subtracting 52 bytes from the total length of the OpenVPN packet. Additionally, the length of the encrypted regular packet is equal to the length of the non-encrypted regular packet; thus, we have found the length of the corresponding regular packet. We confirm our claim by performing tests with our setup. We capture the regular traffic, which is sent to the OpenVPN software, as well as the outgoing OpenVPN traffic. We compare the corresponding packets based on arrival time and calculate the differences in length. We find that the differences are always equal to the fixed value of 52 bytes. A visualisation is shown in Figure 4.3.

If we would change the configuration of our setup to use the IPv6 protocol instead of IPv4, our claim would still be valid. However, the fixed difference in length would be 20 bytes larger, i.e. 72 bytes, due to the difference in header lengths as discussed earlier. On the contrary, the choice of symmetric-key cipher algorithm can possibly break our claim. As with GCM, an algorithm is necessary that preserves the length of the plain text, hence not applying padding. Our claim, therefore, is not specific to the GCM algorithm, but it is to algorithms that preserve the length of the plain text. For instance, the AES-256-CBC [25] algorithm currently acts as the default fallback cipher for OpenVPN [2] and, therefore, might be used in setups where one of the

[2]https://openvpn.net/vpn-server-resources/change-encryption-cipher-in-access-server/ (accessed on 20-05-

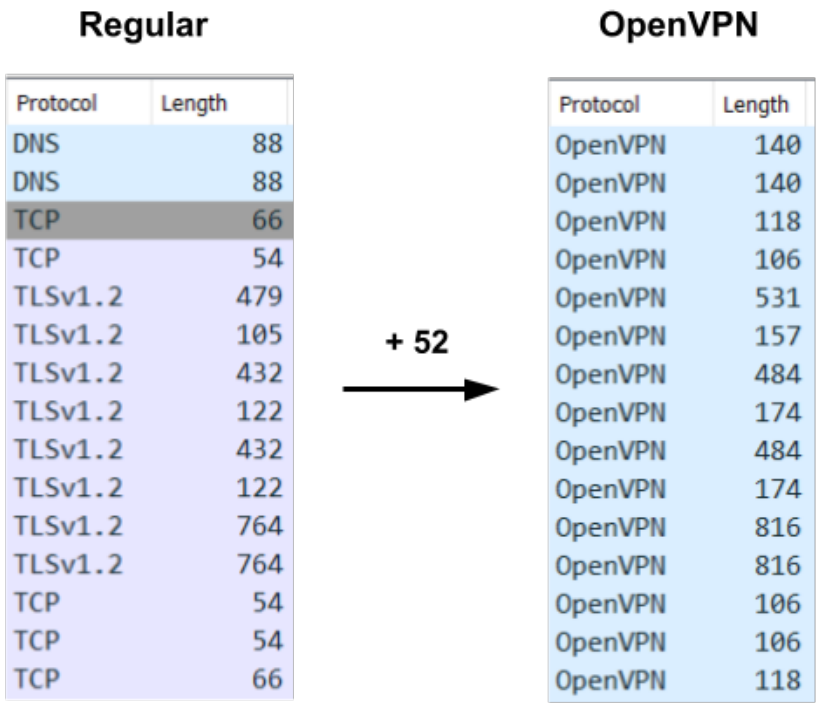


Figure 4.3: Length differences between regular packets and their corresponding OpenVPN packets

endpoints does not support GCM. CBC (Cipher Block Chain) mode applies padding, meaning that our claim would not be valid when a fallback to AES-256-CBC occurs.

4.3 Feature discovery

The previous section discussed how the length of encapsulated regular packets can be deduced from OpenVPN traffic packets. We will now show how this observation can lead to discovering usable features to perform classification. Furthermore, we elaborate on some observations in Section 4.1. The traces that were analysed for feature discovery are an extended version of the web crawler traces used in Section 4.1. The web crawler process is executed two more times, hence yielding three traces per combination of OS, web browser, and traffic type. This extension aims to limit the effect of random noise in (some of) the traces. This way, the probability that random noise affects our findings significantly decreases. This section discusses the discovered features we can use to perform classification. The values that will be reported for a particular combination of classification options are always averaged over the three traces that were captured for that combination.

4.3.1 TCP Acknowledgement

TCP Acknowledgement (ACK) packets can be used to classify traffic types. First, we need a method to identify TCP ACK packets in the stream of OpenVPN packets. To achieve this, we take advantage of our observation in Section 4.2; if we know the length of regular TCP ACK packets, we can identify these within OpenVPN packets. A problem occurs in this situation: a

2022)

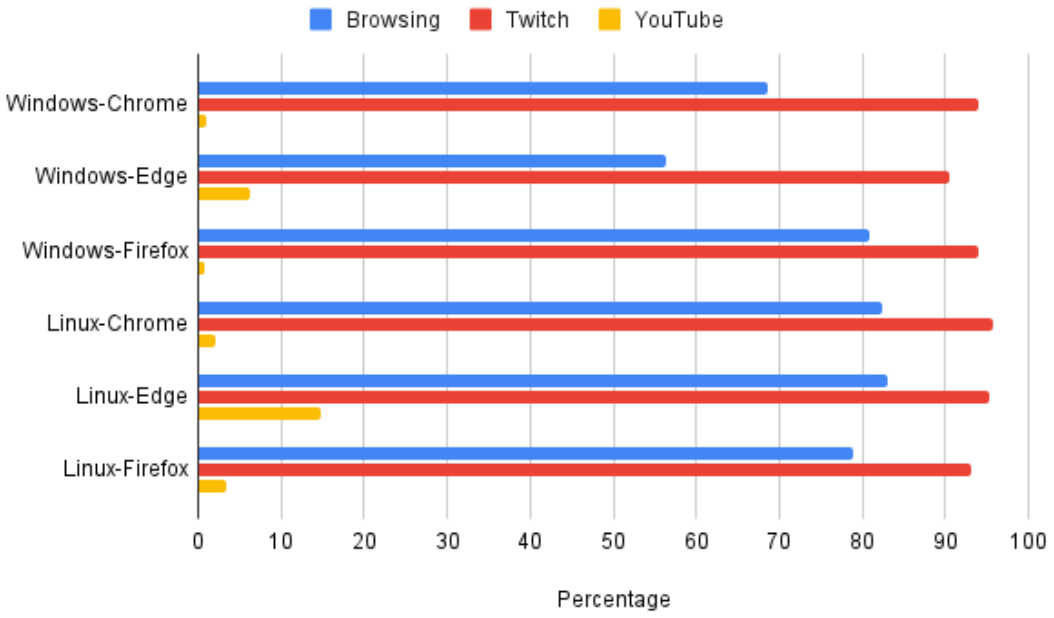


Figure 4.4: TCP ACK percentage of web crawler traces

TCP ACK packet can contain a payload of unknown length, making it difficult to match TCP ACK packets that do carry payload with a certain length. Therefore, we will only focus on the packets that do not carry any payload. These packets, in particular, thus only consist of a set of headers, which have fixed lengths if no additional options are used. Specifically, a TCP ACK without payload and no additional header options will have a length of either 54 bytes or 74 bytes, depending on the used IP protocol (IPv4 or IPv6). However, during analysing regular traffic, we also encounter TCP ACKs without payload containing an additional option with the timestamp in the TCP header. Fortunately, this additional option has a fixed length of 12 bytes. Therefore, we can also consider this specific type of TCP ACK, which will have a length of either 66 bytes or 86 bytes. Knowing that OpenVPN packets are precisely 52 bytes larger, we identify OpenVPN packets with a length of 106 bytes and 126 bytes as TCP ACK packets without payload or options. Similarly, we identify OpenVPN packets with lengths of 118 bytes and 138 bytes as TCP ACK packets without payload but with 12 bytes of options containing the timestamp.

We will now show how the identification of these TCP ACK packets can be used to classify the type of traffic in a trace. More specifically, we will use this mainly to distinguish between YouTube streaming and Twitch streaming traffic. Later in Section 4.3.3, we will show a different approach to differentiate between these two options and browsing.
YouTube utilises an Adaptive Bitrate (ABR) streaming algorithm with the Quick UDP Internet Connection (QUIC) protocol [44, 68] on top of UDP for on-demand streaming, unless the client does not support QUIC. When analysing the generated packet stream of our client while streaming YouTube videos, we observe a flow of QUIC traffic, hence confirming that YouTube applies QUIC/UDP streaming also for our setup. Note that QUIC utilises the transport protocol UDP, which is sometimes blocked by middleboxes. To cope with this issue, QUIC implements a fallback to TCP when the setup of the QUIC connection fails [44]. Consequently, YouTube's streaming algorithm falls back to an alternative version using TCP/TLS instead of QUIC/UDP [12]. Our approach will not take into account this fallback possibility and assumes that the client supports QUIC.
On the other hand, Twitch uses HTTP Live Streaming (HLS) with HTTP/1.1 to deliver the

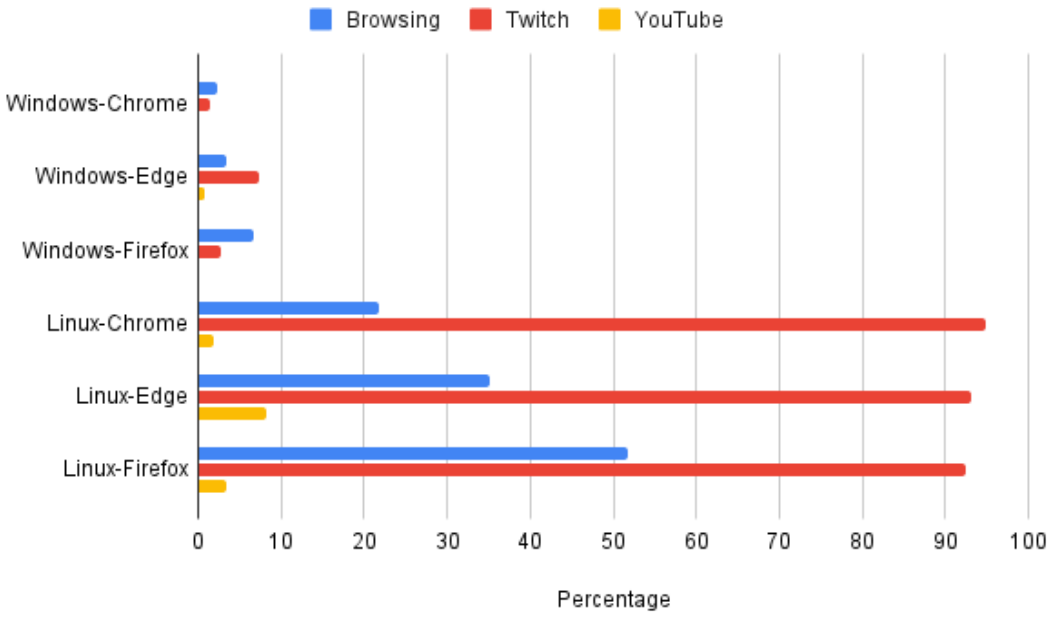


Figure 4.5: TCP ACK, only with extra timestamp option, percentage of web crawler traces

live stream to the viewers [67]. Consequently, this streaming protocol makes use of the TCP transport layer protocol. The essence is that our client generally will be generating QUIC traffic when streaming YouTube and TCP traffic when streaming Twitch.

As mentioned at the start of this chapter, the client is the receiver of the stream, hence generating acknowledgements for the received data of the incoming stream. In Figure 4.4, we show, for every combination of classifications, how many packets in the trace were identified as our targeted TCP ACK (without payload) packets. We represent this fraction as a percentage. We choose to represent it as a percentage rather than an absolute count because the total amount of packets can differ significantly across traces, mainly due to the time span of the trace and the type of generated traffic in the trace.
Clearly, the difference in values between Twitch and YouTube is tremendous. Consequently, we can choose a value that will act as a separation limit for YouTube and Twitch traffic. Based on the figure, we choose the value of 50% as a separation limit; if the observed value is below the limit, we classify the trace as YouTube, otherwise as Twitch. We will thus use this parameter as a feature to separate Twitch traffic from YouTube traffic. Note that, analysing the figure, we can also try to include browsing, hence differentiating between the three traffic classification options using two limits: one limit, say 20%, to separate browsing from YouTube, and another limit, say 90%, to distinguish between browsing and Twitch. However, we think this may introduce more classification errors because the difference between browsing and Twitch is generally not too obvious. Furthermore, the statistics of browsing traffic depend heavily on the executed browsing actions, which can variate a lot. Therefore, we do not implement this additional option.

Besides using TCP ACKs to predict traffic types, we can also use this parameter to make a distinction between the OSes in our setup. Earlier in this section, we mentioned that we occasionally encountered many TCP ACKs including an extra timestamp option in the TCP header. More specifically, this was usually the case when analysing traffic generated by the Linux client. Therefore, we now research if we can use this parameter to separate traffic generated by a Linux client from traffic generated by a Windows client. In Figure 4.5, we show how many TCP ACKs

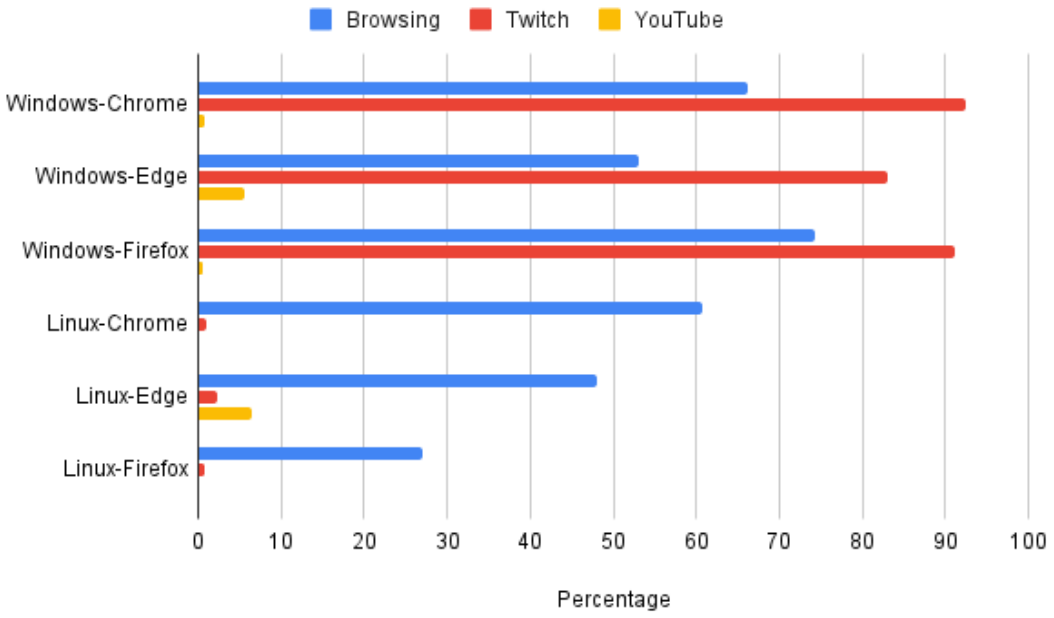


Figure 4.6: TCP ACK, only without extra timestamp option, percentage of web crawler traces

with the extra timestamp option were present for every combination, expressed in percentage compared to the total amount of encountered packets. The difference between this figure and Figure 4.4, which was used earlier in this section, includes that the former only contains the TCP ACKs without payload but with the additional timestamp option, while the latter contains the TCP ACKs without payload both with and without the extra timestamp option. Nonetheless, we see a clear difference between Windows and Linux when looking at the percentages for the TCP ACK with extra timestamp options packet types. This clearly confirms our belief that these packets occur more frequently when using Linux, compared to using Windows. However, the figure shows that the percentage of this packet type for YouTube traffic is significantly smaller than the other traffic types. While the difference between Windows and Linux for YouTube traffic is still substantial, it will be hard to distinguish between Linux and Windows if YouTube traffic is generated, but we are not aware of that since we are focusing on the OSes. One (partial) possible solution to solve this issue includes classifying traffic types prior to classifying OSes. This way, we do know the traffic type prediction, and we can use this to support the OS classification. However, it is very likely that the prediction will not always be correct, so the additional information used for the classification of OSes may actually be false/misleading. Therefore, we choose to classify traces with a relatively high percentage as Linux, specifically with a threshold of 15%, deduced from the figure.

Contrarily to only considering TCP ACKs with the extra timestamp included, we can also look at only TCP ACKs without this option included. While the former was mainly interesting for detecting the Linux OS, we may be able to use the latter to detect the Windows OS. In Figure 4.6, we show how many TCP ACKs without the extra timestamp option were present for every combination, expressed in percentage compared to the total amount of encountered packets. We find that, indeed, we can use this parameter to separate Windows because we observe that only Windows contains values above the threshold of 70%. As a result, we classify traces as Windows if the value of this parameter exceeds the identified threshold.

Our classification approach for the OSes does not cover all scenarios; it is perfectly possible that we can not separate the Linux OS considering TCP ACKs with the extra option nor separate the Windows OS considering TCP ACKs without the extra option. Because there is no obvious way to classify traces in such a scenario, we choose not to classify them at all, hence using an 'Unknown' classification option. The idea behind this is that if we classified traces in this scenario, we would be making guesses without any relatively strong indication of the correct classification. However, we can use another feature to possibly distinguish between the OSes, which will be discussed in Section 4.3.3. Thus, if our classification of OS remains unknown after considering TCP ACKs, it is still possible that we reconsider this classification after analysing the feature in Section 4.3.3.

4.3.2 QUIC Acknowledgement

Similar to analysing the percentage of TCP ACKs in a trace, targeting (and afterwards confirming) the assumption that this value will be significant for Twitch traffic, we can also analyse the percentage of QUIC ACKs because we assumed that this type of network packets would mostly occur in YouTube traffic. We found that lengths of either 81, 82, or 83 bytes can identify an IPv4 QUIC ACK packet without payload, yielding OpenVPN packets of lengths 133, 134, and 135 bytes respectively. Analogously, we need to add the value of 20 bytes to find the same packet types when IPv6 is used instead of IPv4.

In Figure 4.7, we show, for every combination of classifications, how many packets in the trace were identified as our QUIC ACK (without payload) packets. Clearly, there are significant differences across the traffic types. Furthermore, our assumption that YouTube traffic generated by the client contains the most significant percentage of QUIC ACKs compared to Twitch is with this confirmed. Actually, the percentage of QUIC ACKs in the Twitch traces is consistently very close to zero. This is no surprise because the trace should only contain Twitch streaming traffic (apart from some possible background noise) which does not use the QUIC protocol. Because we have found a clear difference between YouTube and Twitch traffic for this parameter, we will use this parameter in our classification process to separate YouTube and Twitch traffic. One could argue that we already use the percentage of TCP ACKs to distinguish between these two, as explained in Section 4.3.1. However, we aim to use this parameter to establish confluence: if both the TCP ACKs and QUIC ACKs parameters yield the same classification result, the confidence that the classification is correct increases. On the contrary, if both yield a different classification result, we can keep in mind that this trace has no obvious classification result, hence a decrease in confidence. Based on the figure, we can also differentiate between browsing and YouTube traffic. We could use this observation in our decision process. However, as discussed earlier, we have to keep in mind that browsing traffic can vary greatly. For example, if we are, by chance, mainly browsing websites that use the QUIC protocol, the percentage of QUIC ACKs in the trace could increase significantly. We can identify browsing traffic based on another feature discussed in Section 4.3.3, which we believe is a better option.

4.3.3 TLS Client Hello

In Section 4.3.1 and Section 4.3.2, we discussed our approach to separate between streaming YouTube and Twitch. However, our traffic classification contains an additional option: browsing. We still need a way to distinguish between browsing and streaming either YouTube or Twitch. We will now show our approach to tackling this.

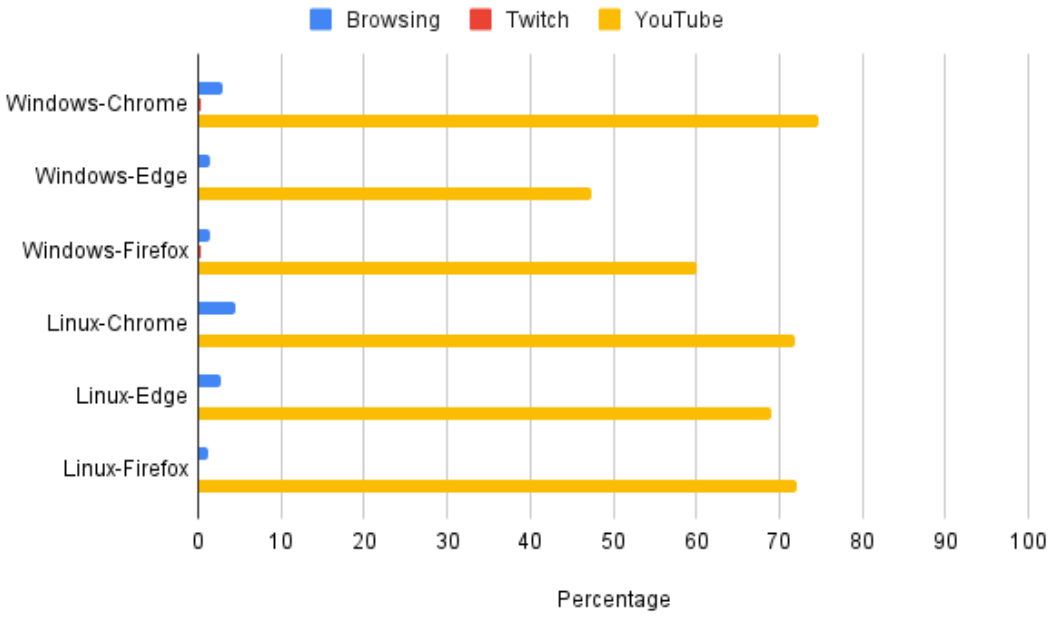


Figure 4.7: QUIC ACK percentage of web crawler traces

When a client connects to a certain domain via HTTPS, a TLS handshake is performed between the client and the server of that domain. This handshake is initiated with the TLS Client Hello (CH) packet sent by the client. We will not go into further detail concerning the TLS handshake since that would be out of context. The critical part here is that every time a client initiates a new HTTPS connection, this handshake is performed, hence sending a TLS CH to the corresponding server. If we are able to identify TLS CH packets in a stream of OpenVPN traffic, we can get an idea of how many distinct HTTPS connections the client is initiating. We can use that information to predict the traffic type that is being generated, more specifically to distinguish between browsing and either streaming YouTube or Twitch. The rationale behind this idea is that when a client is streaming YouTube or Twitch, it generally exchanges data over an already established connection with the server that is delivering the stream, hence initiating few new (HTTPS) connections with other domains. Contrarily, when a client is browsing and thus frequently accessing new webpages or websites, it repeatedly establishes HTTPS connections with new domains to retrieve resources required by the accessed webpages. This impression allows us to differentiate browsing from streaming YouTube or Twitch by considering the amount of generated TLS CH packets sent by the client. We will now show how to identify TLS CH packets in an OpenVPN packet stream and analyse the statistics of sent TLS CH packets per type of traffic, verifying our claim.

To identify TLS CH packets in the OpenVPN stream, we will again utilise our observation in Section 4.2. Therefore, we need to match specific packet lengths with TLS CH packets. Only the TLS layer of the packet is considered when calculating the length of TLS CH. At first glance, the length of a TLS CH seems complicated to determine because it usually contains many different extensions that vary across different TLS CH packets. Still, one key characteristic allows us to match a considerable amount of TLS CH packets with specific sizes. This characteristic includes that if the length of the TLS CH is between 256 and 511 bytes (inclusive), a padding extension is used to extend the length of the packet to at least 512 bytes [43]. This padding extension is being used by all considered web browsers [3] [4]. A visualisation of the padding extension is shown

[3]https://bugzilla.mozilla.org/show_bug.cgi?id=944157 (accessed on 25-05-2022)
[4]https://codereview.chromium.org/62103003 (accessed on 25-05-2022)

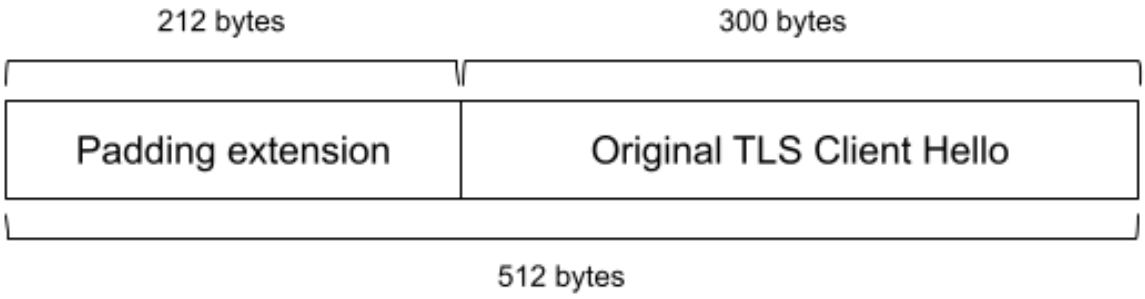


Figure 4.8: TLS Client Hello padding example

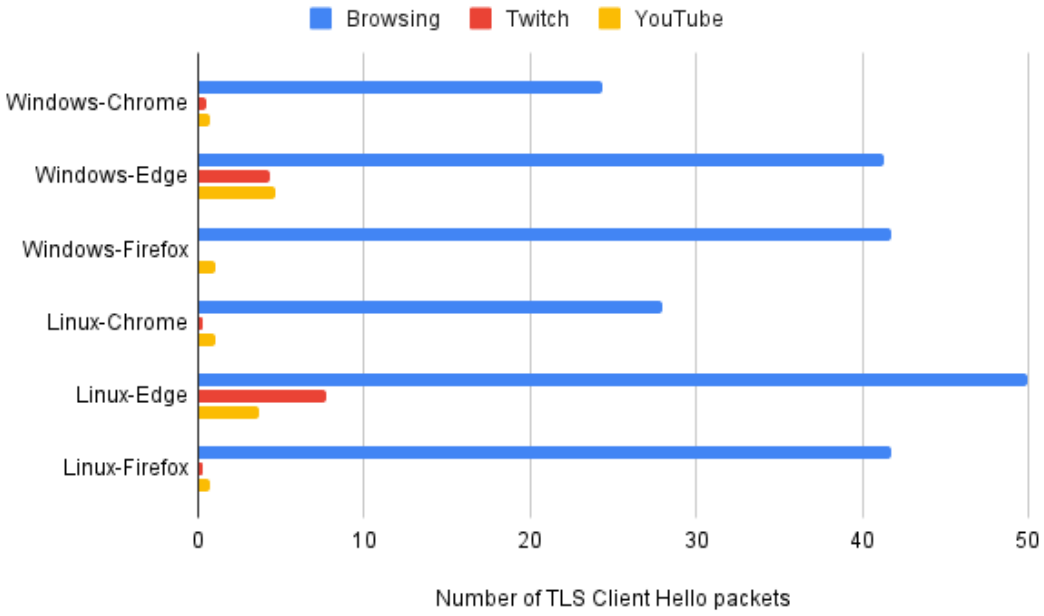


Figure 4.9: TLS Client Hello requests of web crawler traces

in Figure 4.8. The extension is implemented because some TLS implementations running on servers incorrectly interpret TLS CH messages between 256 and 511 bytes as an SSLv2 connection and therefore interrupting the connection [26]. As a result, the padded TLS CH packets will have a length of 512 bytes, allowing us to use this size to identify TLS CH packets in the OpenVPN stream. Clearly, this will only allow us to identify the TLS CH packets that initially had a length between 256 and 512 bytes, so we will only be able to spot this part of generated TLS CH packets. Still, we will show that this is a very interesting observation that we can use for traffic classification prediction. Five more bytes of header values are added to complete the TLS layer section of the packet. Additionally, to complete the full network packet, a set of TCP, IP, and Ethernet headers is added. Consequently, the complete regular TLS CH that we are targeting will have a length of 571 bytes for IPv4 and 591 bytes for IPv6, yielding OpenVPN packet lengths of 623 and 643 bytes respectively.

To measure the amount of TLS CH packets in a trace, we choose to count the number of packets and normalize this for 1 minute instead of using percentages because we observed that the percentage values for this packet type are generally very low. The duration of 1 minute will be the time period of every trace in our dataset upcoming in Section 4.5.1; we discuss this in further

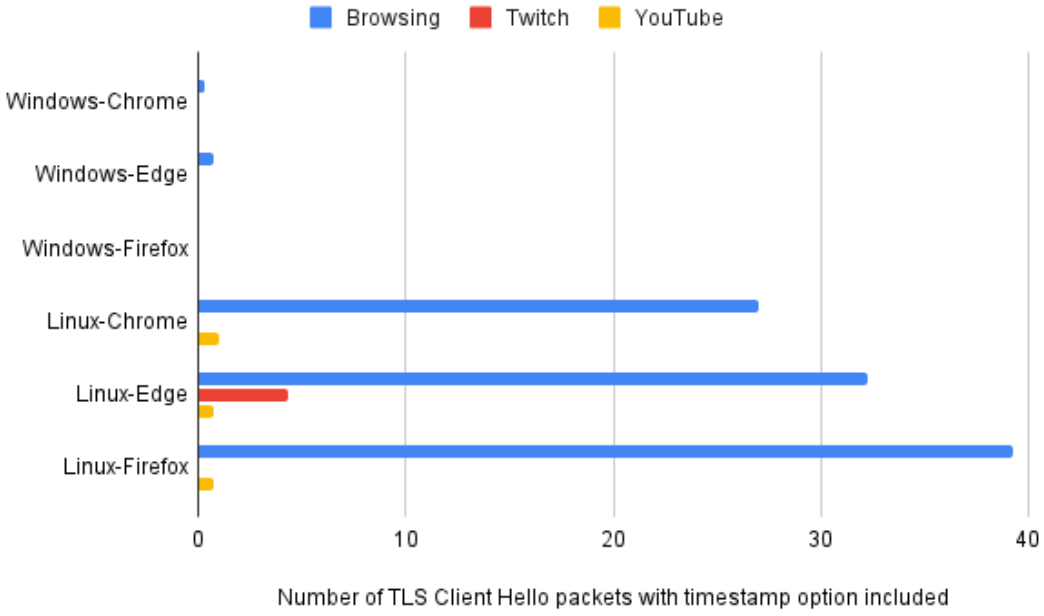


Figure 4.10: TLS Client Hello requests, only with extra timestamp option, of web crawler traces

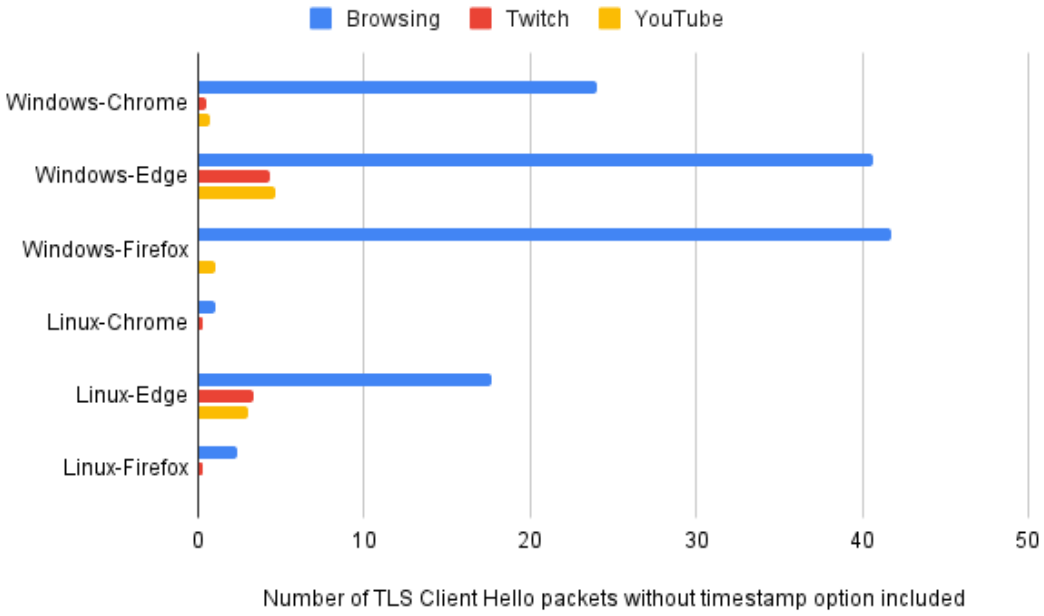


Figure 4.11: TLS Client Hello requests, only without extra timestamp option, of web crawler traces

detail in that upcoming section. In Figure 4.9, we show, for every combination of classifications, how many packets in the trace were identified as TLS CH packets. For every combination of OS and web browser, the number of TLS CH packets is significantly higher for browsing than for streaming YouTube and Twitch, hence confirming our claim that browsing traffic generates the most TLS CH packets. Therefore, we can use this parameter to separate browsing traffic from streaming YouTube and Twitch. Based on Figure 4.9, we choose a threshold of 15 packets to separate browsing from Twitch & YouTube.

A TLS CH packet is constructed using TCP as the transport layer protocol. Therefore, as extensively discussed in Section 4.3.1, we can also examine the presence and absence of the extra timestamp option in the TCP header for TLS CH packets to predict the OS. Contrarily to TCP ACKs, we will use absolute counts for TLS CH packets with and without the extra option. Figure 4.10 and Figure 4.11 show the number of TLS CH packets with the extra option and the number of TLS CH packets without the extra option respectively. For both figures, we see clear differences between the OSes. However, the differences are only significant when the traffic type is browsing, as expected since we argued earlier that a high number of TLS CH packets only occurs for the browsing traffic type. Based on the figures, we classify the OS as Linux if at least a normalized number of 5 TLS CH packets with timestamp options are detected. Similarly, we classify the OS as Windows if at least a normalized amount of 20 TLS CH packets are detected without a timestamp option.

4.3.4 Packet rate

Another parameter we will use as a feature is the packet rate of the trace. We express the packet rate of a trace as the average number of packets per minute of that trace. The packet rate is shown in Figure 4.12 for every combination. We can deduce from the figure that there is no obvious difference between the OS or web browser classifications. However, we do see a considerable difference between the traffic types, mostly between Twitch and YouTube. We thus can use this feature to separate YouTube and Twitch traffic. Looking at the figure, we decide to use the value of 6,000 packets per minute as a threshold to differentiate between these two options. We choose this value because it is clearly both significantly higher than the highest YouTube value and significantly lower than the lowest Twitch value. Note that this is not the exact value the threshold needs to be; one could argue that 5,000 or 7,000 is a better choice. For this classification to be useful, we first need an indication that the captured traffic is streaming traffic rather than browsing. We can achieve this by first considering the TLS CH feature as discussed in Section 4.3.3. Moreover, how we will exactly benefit from this feature will be covered in Section 4.4.

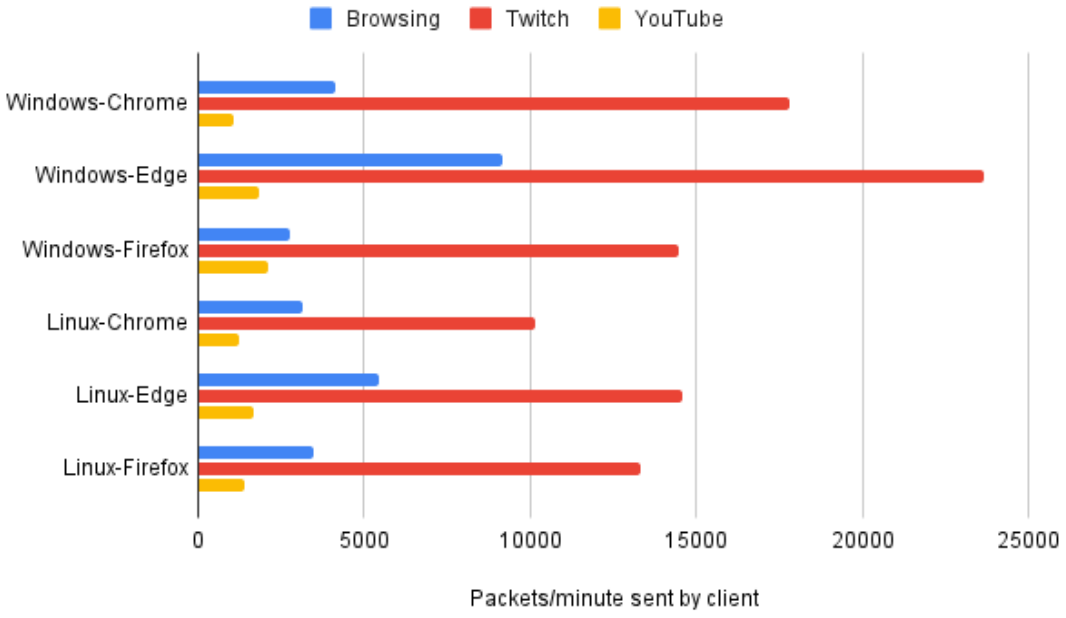


Figure 4.12: Packet rate of web crawler traces

4.3.5 IP time to live

When thoroughly inspecting the OpenVPN network packets sent by the client, we notice that the time to live (TTL) value of the IP header differs between the two OSes. Note that this was observed for the IPv4 protocol. The IPv6 protocol header contains a similar field named 'hop limit'. This field serves exactly the same purpose as the IPv4 TTL field. Therefore, when mentioning the IP TTL value, we refer to the IPv4 TTL as well as the IPv6 hop limit. As discussed in Section 2.4.4, an eavesdropper can inspect these header contents because only the application layer payload (i.e. the encrypted regular packet) of the outer packet is encrypted. We find TTL values of 128 and 64 for Windows and Linux respectively, indicating that the TTL value may depend on the OS. From Section 2.4.1, we can confirm that the OS is responsible for adding the packet headers of the outer packet, including specifying the TTL value in the IP header. Moreover, we find that the specific Windows and Linux OSes used in our setup have different default value settings for the TTL value. This can be confirmed by looking at the settings, online documentation (Windows) [5] [6] or even the source code (Linux) [7] [8]. In addition, the difference in TTL value has been used in related work focusing on OS fingerprinting; [69] classify the values of 64 and 128 as Ubuntu (which is also our Linux OS as mentioned earlier) and Windows respectively. Furthermore, [13] found that Android and iOS use 64 by default, implying that a TTL value of 128 is very unlikely to be generated by these OSes. Besides, they found that most Windows IPv4 packets contained a TTL of 128. We can use this information to distinguish between Windows and Linux by looking for a TTL value of 128 in a trace; if we encounter at least one TTL value greater than 64, we classify the trace's OS as Windows, otherwise as Linux. We choose to check for a value greater than 64 rather than an exact value of 128 because there is a possibility that the MITM intercepting the traffic is multiple hops removed from the client. Of course, because our setup only contains two OSes, we can classify the OS as Linux if no TTL value greater than 64 is found. However, if we chose not to limit the included OSes to only Windows and Linux, we could classify a TTL value greater than 64 as Windows and another TTL value as 'not Windows'.
Until now, concerning the classification of OSes, we discussed a fingerprinting approach to separate the classification options by deducing packet types based on the observed packet sizes. This TTL feature allows us to classify the OS much more effortless. The only element required for this approach is a plain text header value. That is why we can actually consider this as some form of data leakage similar to the topics discussed in Section 3.1.

Unfortunately, this strategy has some drawbacks. If an eavesdropper is at least 64 hops away, the observed value will be smaller than or equal to 64, even if it was 128 at the origin. In that case, one can not benefit from the fact that Windows initially sets a higher TTL value. In addition, we need to keep in mind that changing the default TTL values in the OS settings might affect the method's accuracy.

4.3.6 Maximum packet size

The final parameter that we will discuss is the maximum packet size of a particular trace. In Section 4.1, we found that the maximum packet size is larger for a Firefox client (1451 bytes) than for a Chrome or Edge client (both 1444 bytes). First of all, it is notable that the value is

[5] https://docs.microsoft.com/en-us/powershell/module/nettcpip/get-netipv4protocol?view=windowsserver2019-ps (accessed on 20-05-2022)
[6] https://docs.microsoft.com/en-us/powershell/module/nettcpip/get-netipv6protocol?view=windowsserver2019-ps (accessed on 20-05-2022)
[7] https://github.com/torvalds/linux/blob/master/net/ipv4/ipconfig.c (accessed on 20-05-2022)
[8] https://github.com/torvalds/linux/blob/master/net/ipv6/addrconf.c (accessed on 20-05-2022)

the same for Chrome and Edge but not for Firefox. The most likely explanation is the fact that both Chrome and Edge are built on top of Chromium [9], hence sharing a significant part of functionality and therefore behaving similarly in the context of generated network traffic. Moreover, Firefox is implemented independently from Chromium. When inspecting the plain text traffic generated by all browser clients, we notice that the largest packets are typically categorised as QUIC. For the regular QUIC packets (not encapsulated), we find maximum packet lengths of 1399 bytes for Firefox and 1392 bytes for Chrome and Edge. These values are what we expect because, recalling Section 4.2, the difference between the OpenVPN packets and the regular packets is exactly 52 bytes. To avoid IP fragmentation of QUIC packets, Chromium implements the QUIC protocol in such a way that a QUIC packet as a whole (with UDP, IP, and Ethernet header) is not larger than 1392 bytes [10], which is also exactly the maximum QUIC packet length we observed for the web browsers Chrome and Edge. We did not find any documentation concerning the QUIC implementation in Firefox. However, the maintainers of Firefox have likely chosen a slightly (1399 bytes instead of 1392 bytes) larger maximum value for QUIC packet lengths.
This observation enables us to differentiate between a Firefox client and a Chromium-based client (Chrome or Edge), obviously only if the clients are generating QUIC traffic. On the contrary, because the network stack is implemented in Chromium, which is being used by both Chrome and Edge, we suspect that it will be hard to detect features to distinguish between these two web browsers manually.

4.3.7 Recap

By analysing our extended web crawler traces, we have found several parameters that can be used as features to classify the web browser, operating system, and traffic type of a certain VPN traffic trace. In particular, we use the identification of the following packet types as features: TCP ACKs without payload, both with and without the extra timestamp option, QUIC ACKs, TLS Client Hello packets, both with and without the extra timestamp option, the IP TTL value of the tunnel packets, the maximum packet size, and finally the packet rate. Every feature contributes to at least one classification type. Table 4.3 shows an overview of which classification types every feature contributes to. Except for the IP TTL and the maximum packet size feature, the identification of the features was only possible because of the crucial observation in Section 4.2, hence using packet lengths to identify corresponding packet types. The mentioned packet sizes corresponding to the utilised packet types are summarized in Table 4.4.

Feature	**Classifcation**
TCP ACK	Traffic, OS
QUIC ACK	Traffic
TLS Client Hello	Traffic, OS
Packet rate	Traffic
IP TTL	OS
Max. packet length	Web browser

Table 4.3: Overview of which classifications every feature contributes to

[9] https://www.chromium.org/ (accessed on 20-05-2022)
[10] https://www.chromium.org/quic/ (accessed on 20-05-2022)

Packet type	IPv4 packet		IPv6 packet	
	Size (B)	**OpenVPN size (B)**	**Size (B)**	**OpenVPN size (B)**
TLS CH	571	623	591	643
TLS CH inc. option	583	635	603	655
TCP ACK	54	106	74	126
TCP ACK inc. option	66	118	86	138
QUIC ACK	81, 82, 83	133, 134, 135	101, 102, 103	153, 154, 155

Table 4.4: Packet sizes of packet types concerning discovered features

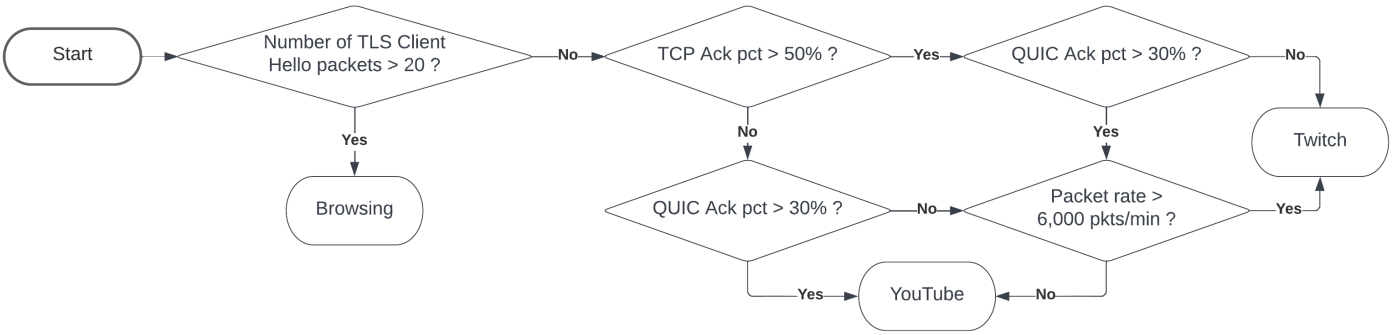


Figure 4.13: Classification process for traffic type (Created in Lucidchart, www.lucidchart.com)

4.4 Classification process

In Section 4.3, we extensively discussed the features we will use for classification. In this section, we describe how exactly we will use these features to obtain a web browser, operating system, and traffic type classification for an (unknown) VPN traffic trace. The threshold values that are used in the figures in this section are consistent with the values we discussed in Section 4.3.

The classification of traffic type is performed as shown in the flowchart in Figure 4.13. First, a check is performed for the amount of detected TLS CH packets. If this amount is significant, the traffic type will be classified as browsing, and the process ends. Otherwise, the significance of the amounts of TCP ACK and QUIC ACK values is considered. Because we are checking two elements with two possible outcomes for each element (significant or not), the check has four possible outcomes. If only one element is significant, we will assign the traffic type corresponding to that element, i.e. Twitch for TCP ACKs and YouTube for QUIC ACKs. If both elements are significant, we assume that the classification is either Twitch or YouTube, but it is not clear which one of these to choose. That is why, in that case, we will consider the packet rate to distinguish between Twitch and YouTube. The last scenario that can occur is when both elements are not significant. We interpret this scenario as an indication that the traffic type is neither Twitch nor YouTube. We suspect this scenario to be browsing traffic where there was little activity and, consequently, insufficient TLS CH packets. Therefore, we classify the traffic type for this scenario as browsing.

Similar to traffic type classification, we can also express the classification of web browser type as a flowchart. This is shown in Figure 4.14. As we argued in Section 4.3.6, we have found no clear feature to distinguish Chrome and Edge because both are Chromium-based, hence sharing a similar implementation of the networking stack. That is why we choose to combine these types

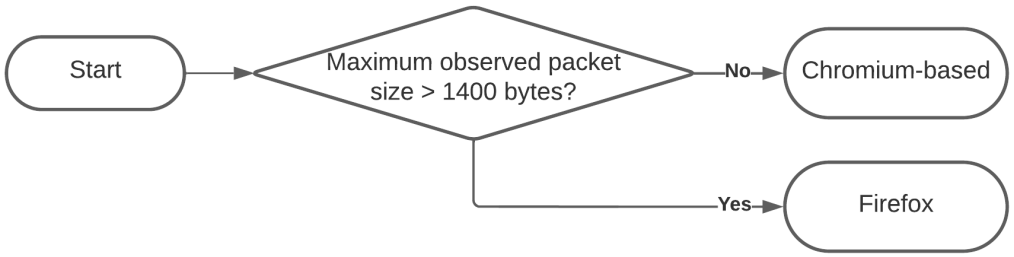


Figure 4.14: Classification process for web browser type (Created in Lucidchart, www.lucidchart.com)

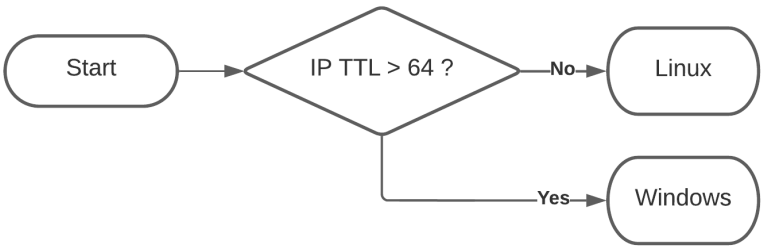


Figure 4.15: Classification process for OS type - approach B (Created in Lucidchart, www.lucidchart.com)

into one type called 'Chromium-based'. Therefore, we will use the maximum packet size feature to distinguish between Firefox and Chromium-based web browsers since we found that a larger maximum packet size is often observed for Firefox.

Finally, the classification of OS type is performed using two different methods. We refer to these two methods as 'approach A' and 'approach B'. We now elaborate on approach A. This approach is slightly different compared to the previous ones as it is hard to express the used method in a flowchart. In Section 4.3 we indicated that we could perform OS classification by analysing the extra timestamp option in the TCP header. We can do this for both the TCP ACK and TLS CH packet types. However, we also mentioned that scenarios could occur where no observation in favour of any classification option can occur, hence classifying the OS as 'unknown'. Therefore, we also include this 'unknown' classification option in the OS classification process. To choose the most suitable classification option, we will keep track of a score for both the Windows and Linux options. For every observation discussed in Section 4.3 that favours a particular OS option, we check if this occurs and increment the score of the corresponding OS option if so. After that, we check the equality of the scores; if they are equal, we classify the OS as unknown. In the other case, we choose the classification option with the highest score. The observations that will be checked and the corresponding favouring OS option are summarised in Table 4.5.
As discussed in Section 4.3.5, we can also classify OS in a completely different way by analysing the IP TTL value in the plain text IP header of tunnel packets. We refer to this method as approach B. The classification process for approach B is shown in Figure 4.15.

Observation	OS option
TCP ACK with option pct $\geq$ 15%	Linux
TCP ACK without option pct $\geq$ 70%	Windows
TLS CH with option rate $\geq$ 5/min	Linux
TLS CH without option rate $\geq$ 20/min	Windows

Table 4.5: Observations per OS option

4.5 Evaluation

Up until now, we have described the workings of our manual approach for classification and we can only assume that our method works. This section aims to properly evaluate our approach. We will describe the dataset that is used for evaluation, the method of evaluation, and finally the results.

4.5.1 Dataset creation

To evaluate our constructed manual approach properly, we need labelled data. In the past, efforts have already been made to create publicly available labelled datasets containing VPN traces [30,65,66]. However, these datasets were all created multiple years ago, meaning they may not entirely represent network traffic as it typically looks nowadays. Network traffic is a very dynamic and evolving field; thus, a span of multiple years is a relatively long period. Moreover, other datasets are obviously not created for our specific use case, hence not containing all the labels required to perform tests. Because of these drawbacks, we conclude that these existing datasets are unsuitable for our use case. That is why we choose to create a new dataset. In the next part of this section, we describe the process of creating the dataset and the exact content of the dataset.

All of the traces in the dataset are captured using the 'pyshark' module [11]. Moreover, the traces are saved in the PCAP Next Generation Capture File Format (pcapng) [71].
For all possible combinations of OS, web browser, and traffic type (e.g. Windows-Chrome-Browsing), we capture a trace with a time length of 60 minutes. As there are 18 possible combinations, this yields a total of 1,080 minutes. Additionally, every trace is split into parts so that every part has a fixed duration. We aim to keep this duration relatively low because this yields more distinct traces, hence more data samples. However, we also need enough traffic in one sample to properly perform classification prediction. We found that a duration of 1 minute is a good trade-off between these two elements. Therefore, every trace is split into samples with a duration of 1 minute. Thus, the dataset contains a total of 1,080 traces. This corresponds to 360 traces, 360 traces, and 540 traces for every web browser option, traffic option, and OS option respectively. Each trace is labelled with exactly one OS, web browser, and traffic option. After completion, the traces containing less than 50 network packets were considered idle and removed.
Concerning the generation of traffic for the different traffic types, generating YouTube and Twitch traffic is relatively easy because this requires one simply to open a stream and start capturing; the client requires no further action. On the other hand, automating the generation of traffic for the browsing traffic type is more complex because this requires the client to interact with the web pages frequently. We automated this process by utilising the Noisy project of Itay Hury, available on GitHub [12]. We made changes to the implementation to fit the needs of our

[11] https://github.com/KimiNewt/pyshark (accessed on 20-05-2022)
[12] https://github.com/1tayH/noisy (accessed on 20-05-2022)

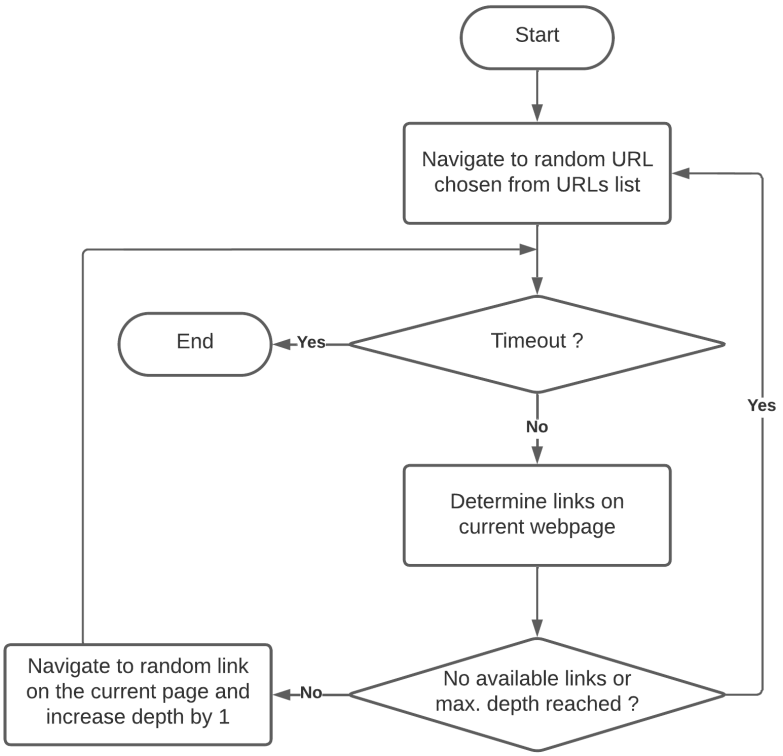


Figure 4.16: Automated process for generating browsing traffic (Created in Lucidchart, www.lucidchart.com)

specific use case. The Noisy process requires a list of URLs as input. These URLs are referred to as root URLs. The process will repeatedly navigate to a randomly chosen root URL and start to follow links on the webpage it is currently visiting. We choose the Tranco [57] top 50 websites to represent this list. A high-level overview of how the Noisy process works is shown in the flowchart in Figure 4.16. Note that the timeout mentioned in the flowchart is set to 60 minutes, as discussed earlier in this section.
The used web browser versions for Chrome, Edge, and Firefox are 97.0.4692, 97.0.1072.62, and 96.0.1 respectively.

4.5.2 Method

We now describe how we evaluate our manual approach with the dataset we created in Section 4.5.1. For every data sample in the dataset, representing a labelled trace with a duration of 1 minute, we independently predict the classification for every classification type. For example, considering a random trace in the dataset, we will make a prediction for OS, web browser, and traffic type independently and successively. After that, the predictions are compared to the actual values and stored. Concretely, the following steps will be performed for every sample in the dataset:

1. Calculate parameter values necessary for classification, e.g. packet rate, TCP ACK percentage, etc.

2. Predict classifications by applying the classification processes described in Section 4.4

3. Store the predictions and true labels for calculation of metrics and confusion matrices later

The first metric used for evaluating the predictions is accuracy. We calculate the accuracy by using the following formula:

$$Accuracy = \frac{\#\ correct\ predictions}{\#\ total\ traces}$$

Thus, the accuracy represents the fraction of the total samples that were correctly classified. Furthermore, the accuracy will be calculated for every classification process separately since classification processes are performed independently. The accuracy for approach A of OS classification will be calculated in two different ways: the first includes the unknown option and considers every unknown classification as false. The second one skips the unknown classification option, hence only considering the cases where either Windows or Linux was predicted.
The calculated accuracy covers a classification type as a whole. However, we also want to gather information about specific classification options within a type. That is why, besides the accuracy of a classification type, we consider the precision, recall, and F1-score for every individual option. To clarify, we define the following terms for a particular classification option:

- **True Positive (TP)**: the number of predictions where the classifier correctly predicts the option (the actual label of the trace is the option)
- **False Positive (FP)**: the number of predictions where the classifier incorrectly predicts the option (the actual label of the trace is another option)
- **False Negative (FN)**: the number of predictions where the classifier incorrectly predicts the actual option as another option
- **Precision**: indicates what fraction of predictions as that option were actually correct.
- **Recall**: indicates what fraction of all samples containing that option were correctly predicted
- **F1-score**: combines precision and recall into a single value, representing the harmonic mean.

The confusion matrix for every classification type is included because this element is required to calculate the TP, FP, and FN values for every classification option. The precision, recall, and F1-score values are expressed in percentages and specifically calculated using the following formulas:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = 2 \times \frac{precision \times recall}{precision + recall}$$

4.5.3 Results & Discussion

The accuracy results are shown in Table 4.6. Because the OS classification contains a special ‘unknown’ result in addition to the ‘correct’ and ‘incorrect‘ prediction results, we included the pie chart in Figure 4.17 for clarity. We observe a high accuracy for OS classification when

Classification	Accuracy
OS - approach A - 'unknown' excluded	97.36 %
OS - approach A - 'unknown' included	54.70 %
OS - approach B	100 %
Web browser	87.96 %
Traffic type	90.52 %

Table 4.6: Accuracy rate per classification scenario

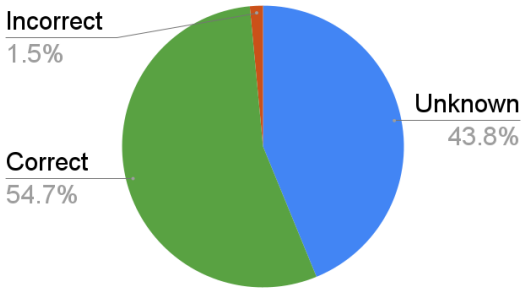


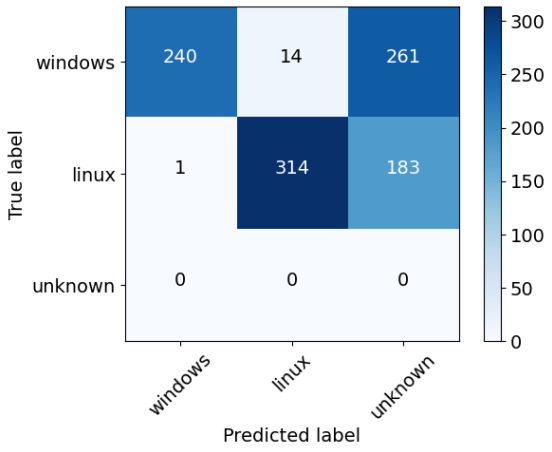
Figure 4.17: OS approach A accuracy pie chart

excluding the unknown option, meaning that the predictions are accurate if the classifier does decide to choose a real classification option. However, when we include the unknown option, we see that this option is chosen in 43.8% of all predictions. As a result, when considering the unknown option as incorrect, the accuracy of the approach drops over 40% (97.36% to 54.70%). Approach B for OS classification yields a perfect accuracy of 100%. This is what we expected since the OS itself sets the IP TTL plain text header value. Therefore, the same value should always be observed unless one modifies the IP TTL value specification in the OS's settings.

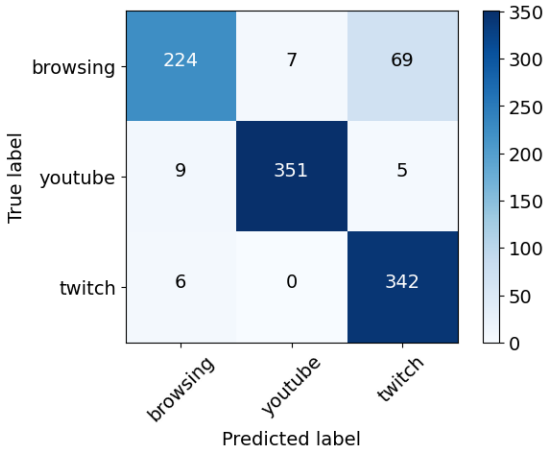
The precision, recall, and F1-score metrics for every classification option are shown in Table 4.7. Moreover, the confusion matrices from which the TP, FP, and FN are deduced are included in Figure 4.18. For OS approach A, both with and without the 'unknown', we see high precision for both options. However, recall is significantly lower when the 'unknown' option is included and considered an incorrect prediction. Considering the two options, the F1-score for Windows is the lowest of the two since precision is just slightly better for Windows, but recall is clearly worse. Actually, the F1-score for Windows is also the lowest across all options.
For OS approach B, both options achieve the maximum score, which is no surprise because earlier in this section, we reported an accuracy of 100% for this approach.
Considering web browsers, it is remarkable that Firefox achieves a maximum precision value, meaning that every time the Firefox option is predicted for a trace, the prediction is correct. This confirms our claim in Section 4.3.6 that a larger packet size is observed for Firefox web browsers compared to Chromium-based web browsers. The maximum recall value for the Chromium-based option also confirms this because this shows that all Chromium-based traces were predicted correctly. The downside of web browser classification is clearly visible in the relatively lower Firefox recall value, indicating that only 63% of all Firefox traces were predicted as Firefox. This suggests that Firefox does not always produce the larger packet size we look for.
Finally, looking at the traffic classification results, the performance values for the YouTube option are the most significant. Looking at the confusion matrix, the primary error clearly represents the prediction of Twitch while the actual option is Browsing. This error occurs for 23% of the total Browsing traces, which explains the relatively low recall value for the Browsing option and precision value for the Twitch option.

		Precision	Recall	F1-score
OS - approach A - 'unknown' excluded	**Windows**	100	94	97
	Linux	96	100	98
OS - approach A - 'unknown' included	**Windows**	100	47	64
	Linux	96	63	76
OS - approach B	**Windows**	100	100	100
	Linux	100	100	100
Browser	**Chromium based**	85	100	92
	Firefox	100	63	77
Traffic	**Browsing**	94	75	83
	YouTube	98	96	97
	Twitch	82	98	89

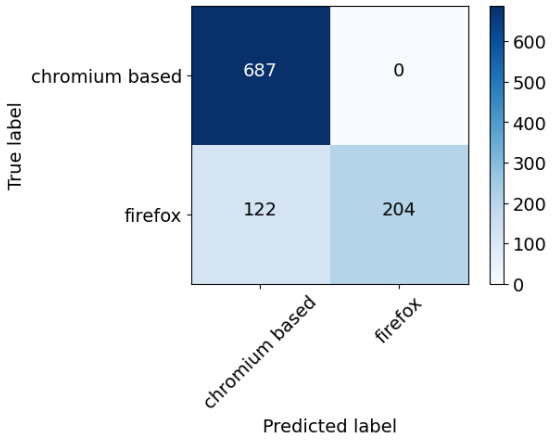
Table 4.7: Precision, recall, and F1-score metrics for all classification options



(a) OS approach A confusion matrix



(b) Traffic confusion matrix



(c) Web browser confusion matrix

Figure 4.18: Confusion matrices for all classification types

4.6 Live classifying tool

To test our approach in a live environment, we created a tool that makes live classification predictions for intercepted OpenVPN traffic. The predictions are made following the classification process described in Section 4.4. The tool is written in Python 3 [73], specifically version 3.9. A simple GUI, created with the Tkinter library [13], is embedded to increase the ease of use. The GUI is shown in Figure 4.19. Some editable configuration parameters are shown on the left side. The source IP address, interface, and VPN server port are necessary to capture the relevant OpenVPN packet stream we are targeting to fingerprint. Note that specifying the source IP address allows us to fingerprint OpenVPN traffic from our device, as well as from another endpoint if we are in a MITM position between that endpoint and its VPN server. The prediction interval specifies the time length (in seconds) of the chunks for which predictions are made. For example, if the prediction interval is 60 seconds, the tool will repeatedly capture a 60-second chunk of traffic and predict classifications for that chunk. The IP TTL checkbox specifies whether the IP TTL value should be used to classify the OS. This allows us to test both methods for classifying the OS as described in Section 4.4.
The upper right pane of the GUI allows us to toggle the process of capturing traffic and performing predictions for the captured traffic chunks. If we hit 'Start', this process will be executed continuously until the 'Stop' button is hit.
The lower right pane shows the prediction results for every captured chunk. Every line indicates the time interval of the chunk that is used for prediction and a classification prediction of every classification type, i.e. OS, web browser, and traffic type.

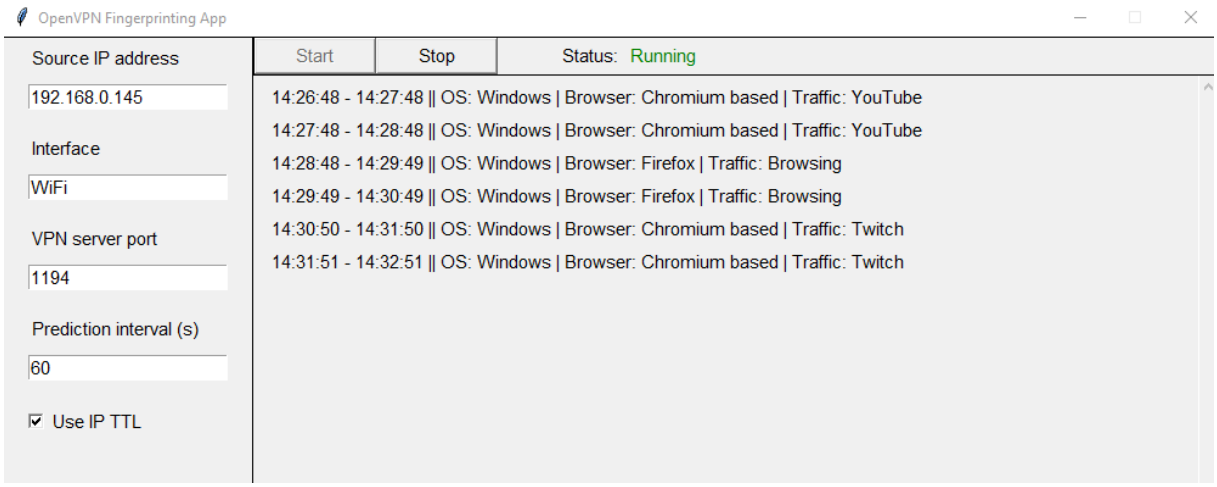


Figure 4.19: Live classifier GUI

4.7 Mitigations

In this section, we will discuss some steps a user can take to counter the fingerprinting method used in this chapter, hence lowering the accuracy of our approach.

As discussed in Section 4.2, the observation that we frequently use in the fingerprinting approach is only valid if the regular packets are encrypted without applying padding. Therefore, if we alter the OpenVPN configuration to use a symmetric-key cipher that applies padding, the accuracy of our fingerprinting approach should decrease significantly. If we change the cipher to

[13] https://docs.python.org/3/library/tkinter.html (accessed on 20-05-2022)

Packet type	IPv4 packet		IPv6 packet	
	Size (B)	OpenVPN size (B)	Size (B)	OpenVPN size (B)
TLS CH	571	~~623~~ 628	591	~~643~~ 644
TLS CH inc. option	583	~~635~~ 644	603	~~655~~ 660
TCP ACK	54	~~106~~ 116	74	~~126~~ 132
TCP ACK inc. option	66	~~118~~ 132	86	~~138~~ 148
QUIC ACK	81, 82, 83	~~133, 134, 135~~ 148	101, 102, 103	~~153, 154, 155~~ 164

Table 4.8: Packet sizes of packet types with padding included

AES-256-CBC, which is the default fallback cipher as discussed earlier, a block size of 16 bytes is maintained. As a result, if the packet length is not exactly a multiple of 16 bytes, padding will be applied. Clearly, if this padding step is performed, the lengths of the packet types we are trying to identify will be larger. As a result, we will not be able to identify the correct packet types. To demonstrate the differences in OpenVPN packet sizes, we calculated the packet OpenVPN packet lengths that would be observed if AES-256-CBC is used. The results are shown in Table 4.8. Note that the crossed-out values represent the original packet sizes for the AES-256-GCM cipher used in Table 4.4.
One could argue that to use our fingerprinting approach for a setup that applies padding, we can simply change it by matching the packet types with the adjusted lengths. However, this adjustment introduces two issues. First, as shown in Table 4.8, there will be multiple packet types with the same length, meaning that we can not longer map a unique length to a single packet type. For instance, an IPv6 TLS CH will have an OpenVPN size of 644 bytes, which would be the same length for an IPv4 TLS CH with the timestamp option included. The same applies to an IPv6 TCP ACK & an IPv4 TCP ACK with a timestamp option, and also to an IPv6 TCP ACK with a timestamp option & an IPv4 QUIC ACK. The second issue includes that we would generate a significant amount of false positives because other types of packets with slightly different lengths will also be padded to the same length, hence being identified as our target packet type. Because of these issues, we suspect that the accuracy will still drop significantly even if we adjust our approach to try to cope with padding.

Several defences against WF attacks have been proposed in related work. All of these defences alter the packet size and possibly also packet timing information of the generated traffic stream. Therefore, these defences are also effective against our fingerprinting attack. Dyer et al. [21] discussed multiple methods for applying padding, including the following:

- *Linear padding*: Increase packet size to nearest multiple of 128, or the MTU if the MTU is smaller.
- *Exponential padding*: Increase packet size to nearest power of 2, or the MTU if the MTU is smaller.
- *Mice-Elephants padding*: If packet size is ≤ 128, increase to 128; else increase to MTU.
- *Pad to MTU*: Increase packet size to MTU.

All padding methods have their advantages and disadvantages. When choosing 'Pad to MTU', for instance, the packet size information becomes useless for an attacker because every single packet will have the same length. However, this method also introduces the most significant amount of bandwidth overhead. They also introduce a new mitigation approach called 'Buffered Fixed-Length Obfuscator' or 'BuFLO'. BuFLO sends fixed-length packets at fixed time intervals to hide both packet size and timing information of the traffic stream. The downside of BuFLO,

however, is a significantly inefficient usage of resources.
Wright et al. [75] utilise a technique called 'traffic morphing', which alters the packet size distribution of the generated traffic. In the context of WF, the distribution is altered to look like the distribution of a very frequently accessed webpage, which the authors chose to be "google.com".

Another way to counter the fingerprinting approach is by generating background traffic. Our approach assumes minimal background traffic. If the majority of the traffic consists of background traffic that is not useful for classification, many packet types identified by matching lengths will be irrelevant packets that are part of the background traffic, hence false positives. Furthermore, background traffic will yield higher packet rates, which is an issue for the packet rate feature. Generating background traffic can be done in numerous ways. One could, for example, initiate multiple random audio and/or video streams using separate browser tabs. Another possibility is using the Noisy tool that we adjusted in Section 4.5.1 to create the dataset. The Noisy tool is actually intended to add extra obscurity by generating random HTTP/DNS traffic noise in the background. In addition, Panchenko et al. [53] have also used background noise as a countermeasure for the WF attack they created. They show that the additional traffic generated by one background page leads to a decrease in the classifier's performance.

Our approach is specifically constructed for OpenVPN traffic. Therefore, one could use another type of VPN to defend against the approach. Moreover, one could also use another privacy-enhancing mechanism such as Tor.
Furthermore, the classification options do not cover all possible classes for every classification type. For instance, many more web browsers exist than the three web browsers we chose to handle. Of course, if one chooses to use a web browser that is not in our list of selections, we cannot correctly identify it. The same applies to the other handled classification types.
Moreover, taking web browsers as an example, it may be possible that significantly older versions of web browsers that we do handle also yield a traffic stream with different characteristics since we only focus on the most recent versions of the selected web browsers.

Chapter 5

VPN operating system, browser, and traffic type fingerprinting: machine learning approach

In Chapter 4, we described our approach to classify OpenVPN traffic traces without using ML techniques, which are commonly used in related work as described in Section 3.2. In this section, we will test some of the ML techniques described in Section 3.2 by applying them for our use case, which includes predicting OS, web browser and traffic type based on OpenVPN traces. We find it interesting to test the related work implementations because it allows us to observe how they perform for our use case compared to the use case they were originally intended for.

5.1 Method

The dataset used to evaluate the ML classifiers will be the one we created to test our manual approach in Chapter 4. For the details of the dataset, we refer to Section 4.5.1. We adopt the FlowPic and Var-CNN approach described in Section 3.2. The most important characteristics for both approaches were summarised in Table 3.1. From these characteristics, it is clear that both approaches are very different. First of all, the FlowPic classifier was originally created solely for traffic type classification purposes, which is also included in our use case. However, we will also use this classifier for OS and web browser classification. On the other hand, the Var-CNN classifier was made for website fingerprinting purposes, which is a significantly different use case than ours. Secondly, FlowPic considers VPN traffic while Var-CNN considers Tor traffic. Lastly, although both approaches use timing features, only FlowPic uses packet sizes, and only Var-CNN uses packet directions.
The authors of Var-CNN claim that their approach is data-efficient, hence performing well even with low amounts of data. This is particularly interesting for us because our dataset contains few data elements in an ML context.

The implementation of the FlowPic method is made available by the authors on GitHub [1]. A LeNet-5 style architecture [46] consisting of seven layers is used. The first four layers are represented by two pairs of a convolutional and max-pooling layer. Next, the output is converted to a one-dimensional vector with a flatten layer before being forwarded to a fully-connected layer. Finally, the output layer is the softmax layer, which does not have a fixed size since the size

[1] https://github.com/talshapira/FlowPic (accessed on 20-05-2022)

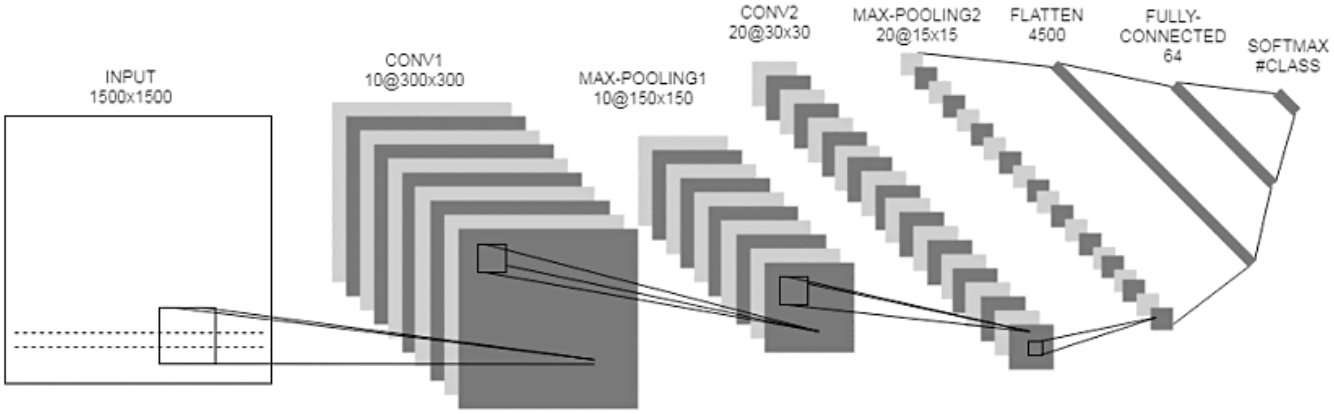


Figure 5.1: Illustration of the FlowPic LeNet-5 style architecture [65, 66]

depends on the number of used classes in the specific classification sub-problem. The activation function 'ReLU' [49] is applied to the output of every convolutional and fully-connected layer. In addition, dropout [70] is used in the second convolution layer and the fully-connected layer. The Adam [40] optimizer is used for the optimization process. The classifier expects the input traces in the shape of a two-dimensional 1500x1500 histogram. Because the traces in our dataset are contained in the pcapng format, we first implement a process to convert a trace from the pcapng format to a 1500x1500 histogram. After that, we split our dataset with converted traces into a training set and test set with proportions of 80% and 20% respectively. Finally, we separate 15% of the training set representing the validation set. The resulting training and validation are used for training the classifier, and the test set is saved for evaluating the classifier later on after the training phase has finished.

The implementation of the Var-CNN method is also available on GitHub [2]. Var-CNN's baseline architecture is based on ResNets [31], which are state-of-the-art convolutional neural networks (CNNs) for image classification purposes. ResNet has multiple variants of different sizes. To minimize training costs, the smallest variant is chosen. This variant consists of 18 layers and is referred to as ResNet-18. The architecture of ResNet-18 is illustrated in Figure 5.2. ResNet-18 has four stages, indicated by a different colour in the figure. The first layer, indicated in orange, is not considered a stage. Each stage contains two convolutional blocks, and a convolutional block is represented by two convolutional layers, yielding a total of four convolutional layers per stage. A feature of this architecture that helps for optimization is a residual 'skip' connection between the input and output of a convolutional block. This is represented in the figure by the arrows together with the plus sign. As the name suggests, skip connections allow a subset of layers to be skipped by adding the skipped part's input to the skipped part's output. In this scenario, we see that every individual convolutional block can possibly be skipped. As for the FlowPic classifier, the Adam optimizer is also used for the optimization process of Var-CNN.
Direction and time metadata features are considered for this classifier. We first need to convert our pcapng dataset files to text files, with each line representing the direction and timestamp of the corresponding packet. This also means that packet sizes are not considered and hence not included in the new data format. Because this classifier focuses on the classification of websites and differentiates between monitored and unmonitored websites, we need to be a little creative to use this for our use case. We first configure the number of unmonitored websites to 0, resulting in the classifier neglecting the part of the unmonitored websites. After that, we set the number of monitored websites to the number of classification options we have. For instance, if we want to classify the traffic type, we set the number of monitored sites to 3 because we

[2] https://github.com/sanjit-bhat/Var-CNN/blob/master/wang_to_varcnn.py (accessed on 20-05-2022)

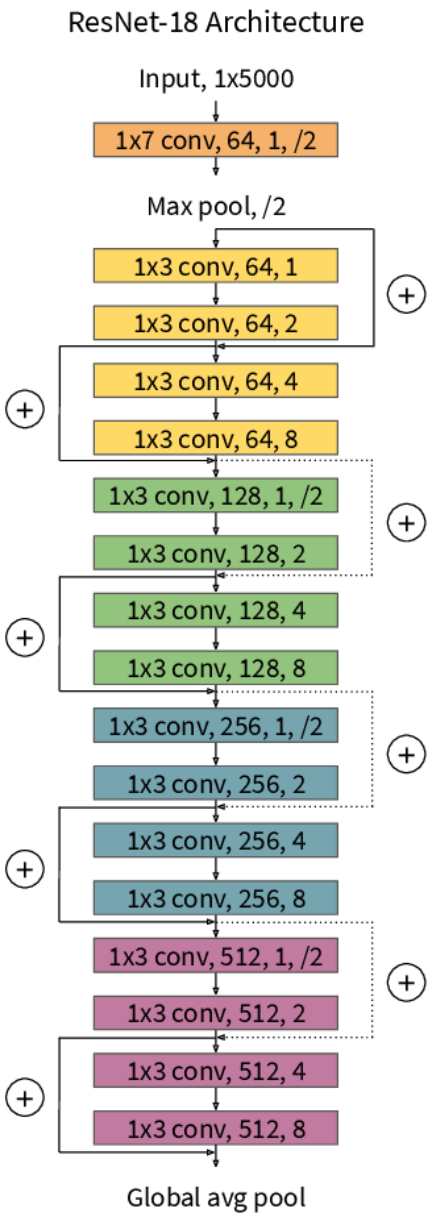


Figure 5.2: Illustration of the Var-CNN ResNet-18 architecture [10]

have three options, i.e. browsing, YouTube, and Twitch. The classifier will try to distinguish between the different monitored websites, yielding the desired results for our use case because the classifier will actually be distinguishing between our classification options.

5.2 Results & Discussion

For both classifiers, the accuracy results are shown in Table 5.1. The precision, recall, and F1-score results are shown in Table 5.2. In both tables we also include the results of our manual approach, described in Section 4.5.3, with the purpose of comparing this with the machine learning classifiers. In addition, the confusion matrices are included in Figure 5.3 and Figure 5.4 for the FlowPic and Var-CNN classifier respectively.
Because our manual approach combines the 'Chrome' and 'Edge' web browsers as 'Chromium based', the classification of web browsers is analysed in two different manners in both tables. The first considers all three options, hence using both the 'Chrome' and 'Edge' options, while

Classification	FlowPic	Var-CNN	Manual
OS	90.28 %	70.75 %	100 %
Browser - 3 options	67.59 %	54.29 %	/
Browser - 2 options	80.42 %	66.43 %	87.96 %
Traffic type	97.69 %	84.29 %	90.52 %

Table 5.1: Accuracy rate per classification type per model

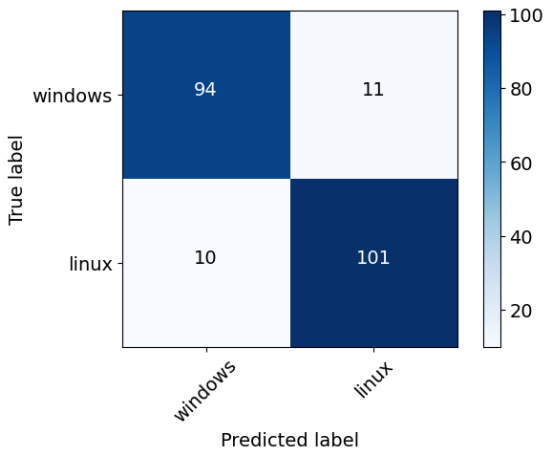
		FlowPic			Var-CNN			Manual		
		P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
OS	**Windows**	90	90	90	81	54	65	100	100	100
	Linux	90	91	91	65	88	75	100	100	100
Browser - 3 options	**Chrome**	58	75	65	52	64	57	/	/	/
	Edge	71	54	61	48	47	47	/	/	/
	Firefox	80	74	77	67	51	58	/	/	/
Browser - 2 options	**Chromium based**	77	88	82	78	46	58	85	100	92
	Firefox	85	72	78	62	87	72	100	63	77
Traffic	**Browsing**	97	96	97	79	86	82	94	75	83
	YouTube	99	100	99	92	97	94	98	96	97
	Twitch	97	97	97	82	70	75	82	98	89

Table 5.2: Precision, recall, and F1-score metrics for all classification options per model

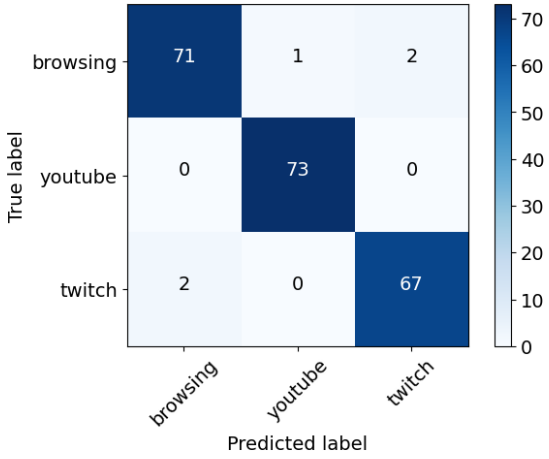
the second combines these options and labels them as 'Chromium based'. Since our manual approach only uses the second manner, no results are reported for the first one. On the other hand, the machine learning classifiers are tested for both cases, yielding an additionally interesting comparison besides comparing the machine learning and manual approach.
Note that in Section 4.5.3, three different scenarios were evaluated for OS classification for the manual approach. For comparison with the machine learning classifiers, we choose only to include the best performing scenario, which is 'approach B'.

For all analysed metrics, the FlowPic classifier clearly outperforms the Var-CNN classifier. One explanation is that the original use case of the FlowPic classifier is more similar to our use case than the Var-CNN classifier because FlowPic focuses on VPN traffic and classification of traffic type while Var-CNN focuses on Tor and website fingerprinting. Besides, it is also possible that packet size features, which are used only by FlowPic, are very effective for our use case.

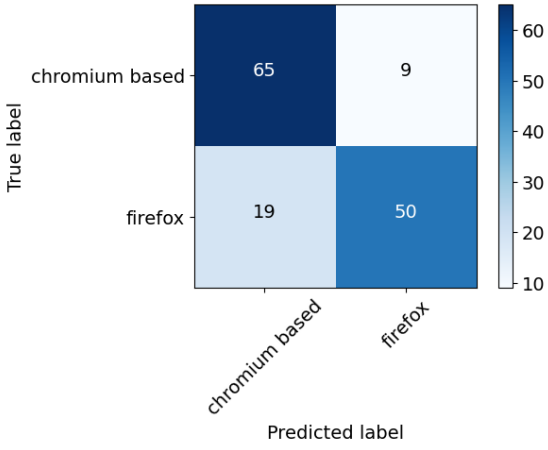
When comparing the best machine learning classifier (FlowPic) with our manual approach based on accuracy, we see that the manual approach performs better for OS and web browser classification with two classes while the FlowPic classifier performs better for traffic classification. The fact that FlowPic performs best for traffic classification is not illogical since this corresponds to the original use case of FlowPic. Let us compare them based on precision, recall, and F1-score values instead of accuracy. We see that the recall and F1-score values for the Firefox web browser are actually slightly better for FlowPic due to the relatively low recall value for the manual approach. Using FlowPic, we can evaluate the performance for the Chrome, Edge, & Firefox web browser classification instead of only combining Chrome and Edge into one option, which we did for the manual approach. Furthermore, the three options' accuracy and F1-scores for the web browser classification yielded decent results for FlowPic. Therefore, FlowPic clearly is the best option for this classification scenario.



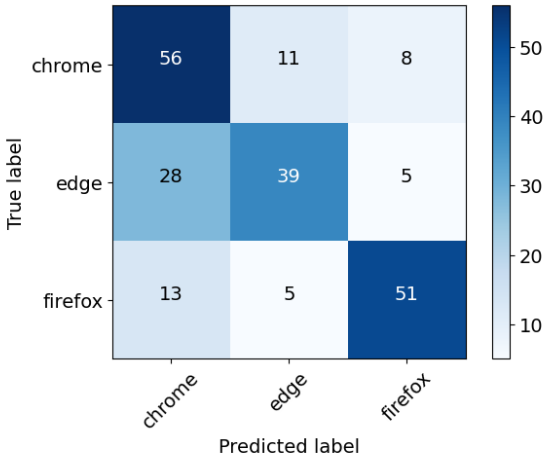
(a) OS confusion matrix



(b) Traffic confusion matrix



(c) Web browser - 2 options - confusion matrix



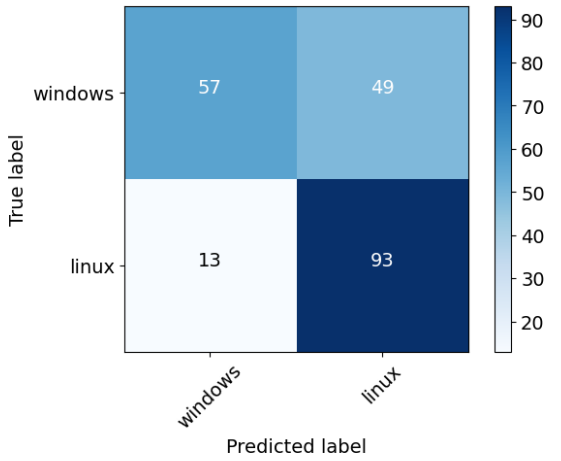
(d) Web browser - 3 options - confusion matrix

Figure 5.3: FlowPic confusion matrices

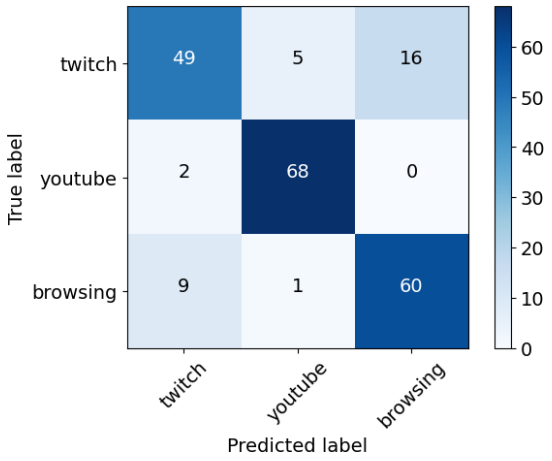
Looking at the different classification types for every classifier, we conclude that web browser classification unquestionably performs worst for all analysed metrics. Considering the F1-scores for web browser classifications of the FlowPic classifier, we see that Firefox's score remains similar. At the same time, Chrome and Edge have relatively low scores if separated but a significantly higher score if combined (Chromium based). This supports our claim that Chrome and Edge are hard to distinguish because both are chromium based as discussed in Section 4.3.6 and Section 4.4. Moreover, when analysing the FlowPic and Var-CNN web browser confusion matrices in Figure 5.3d and Figure 5.4d respectively, we find for both that the most common error includes predicting Chrome while the actual label is Edge. This error is significantly higher than the other way around, i.e. predicting Edge with actual label Chrome.

Observing the F1-scores for traffic classification, we see that YouTube clearly performs best for all classifiers, hinting that the characteristics (timing, direction, sizes) of YouTube traffic clearly differ from the characteristics of the other considered options. Moreover, this also hints that browsing and Twitch traffic have relatively similar characteristics resulting in the classifier

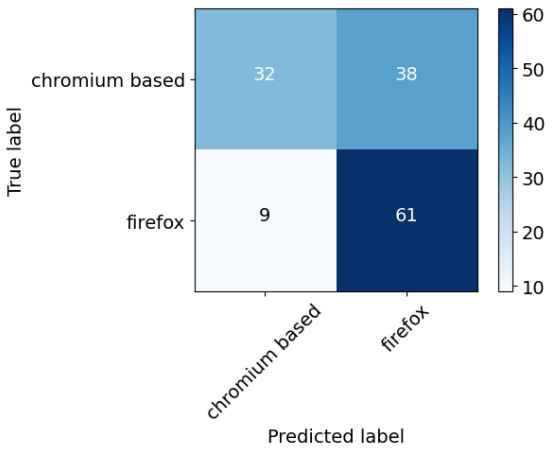
having more trouble distinguishing between the two. This is also visible in the confusion matrices of the Var-CNN and Manual approach shown in Figure 5.4b and Figure 4.18b respectively.



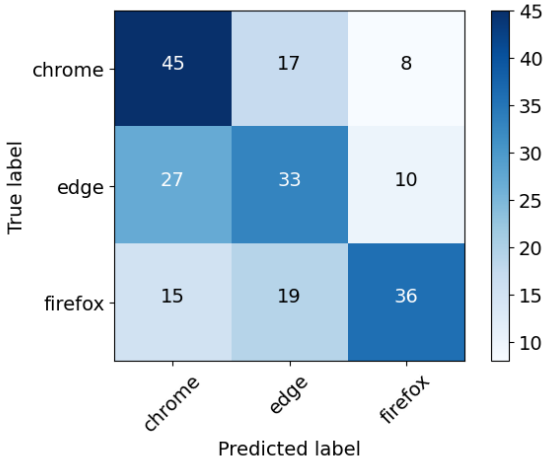
(a) OS confusion matrix



(b) Traffic confusion matrix



(c) Web browser - 2 options - confusion matrix



(d) Web browser - 3 options - confusion matrix

Figure 5.4: Var-CNN confusion matrices

Chapter 6

Conclusion

Nowadays, VPNs are used frequently to provide anonymity and/or privacy, although they were not originally intended for these purposes. In this thesis, we aimed to examine existing and discover new methods that result in privacy leakages of a VPN. Specifically, we focused on fingerprinting techniques that reveal information about user device characteristics, including the operating system, web browser, and traffic type of the user. We developed a manual approach classifier which classifies a trace based on features that were obtained through analysis of traces, without the aid of machine learning techniques. The classifier achieves an accuracy of 100% for classifying the Windows & Linux OSes, 87.96% for classifying the Chromium based & Firefox web browsers, and 90.52% for classifying browsing, YouTube & Twitch traffic. Note that we evaluated multiple scenarios for the OS type, we selected the best performing one. In addition, the lowest observed F1-score for an individual classification option is 77% for the Firefox web browser. Therefore, looking back at the defined research questions, we conclude that it is generally possible to predict a VPN user's device characteristics with our manual approach. There is, however, one pair of web browsers, consisting of Chrome and Edge, for which we did not manage to find a way to differentiate. As a result, we conclude that we cannot distinguish this specific pair of web browsers in our manual approach.
Moreover, in order for our method to work, the scenario needs to comply with some unrealistic assumptions. First, all the classification types do not cover all possible options within a type. For instance, if one chooses to use a web browser that is not in our list of selections, say Safari [1], we are not able to correctly identify it. The same applies to the other handled classification types. That is why we must assume that one of the included options is correct for every classification type. It would have been more realistic if we did not assume that one of the options is correct, hence introducing an additional possibility that all of our considered options are incorrect. A possible way of handling this is by adding an additional 'Other' option for every classification. Secondly, we assume the usage of a desktop browser. We do not know how our method performs in the mobile context since we did not have time to analyse and evaluate this properly. That is why we have to assume a desktop-only scenario. Lastly, we assume that the user is generating a low amount of irrelevant background traffic since this will have an effect on the calculated parameters/features for a trace as discussed in Section 4.7.

We also adopted two state-of-the-art machine learning approaches in the context of classification with fingerprinting techniques based on VPN or Tor traffic. The classifiers were utilised with the collected data used to evaluate our manual approach. The best performing classifier, being FlowPic, achieves an accuracy of 90.28% for classifying the Windows & Linux OSes, 80.42% for classifying the Chromium based & Firefox web browsers, 67.59% for classifying the Chrome,

[1] https://www.apple.com/safari/ (accessed on 05-06-2022)

Edge & Firefox web browsers and 97.69% for classifying browsing, YouTube & Twitch traffic. Consequently, we conclude that it is generally possible to predict the device characteristics of a VPN user by adopting a state-of-the-art machine learning approach.
When comparing the manual approach with the FlowPic classifier, we discussed in Section 5.2 that the manual approach performs better for OS and web browser classification with two classes while the FlowPic classifier performs better for traffic classification and web browser classification with 3 options, based on all analysed metrics. Therefore, when comparing both approache, there is no clear outperformer since the better choice depends on which classification type in particular to consider.

Users can apply some methods to make sure our manual approach classifier's performance drops significantly to protect their privacy. Most of these methods are consequences of the unrealistic assumptions that were described earlier in this chapter. To start, the OpenVPN configuration settings can be altered to use a symmetric-key cipher that applies padding. An example of such a cipher is AES-CBC. Generally, using any padding scheme works as a mitigation against our manual approach. Furthermore, deliberately generating lots of irrelevant background traffic, for instance, uploading a file to a server or using a tool like Noisy [2], can affect the performance of the classifier since it is not able to distinguish between the relevant and irrelevant network packets. Also, making sure that no classification option can be correct by, for example, using a Safari web browser as discussed earlier, is a prominent method to use as a mitigation. Finally, because the fingerprinting method focuses specifically on VPN traffic, another privacy-enhancing technology can be used to provide a higher level of privacy and/or anonymity. A popular alternative is Tor. All of these possible mitigations were also extensively discussed in Section 4.7.

Writing this thesis was not easy at all for me. In the past and throughout writing this thesis, I often had trouble with adequately formulating my thought process in text. I did not know how to properly express in words what I wanted to write down. Because that skill was obviously required to complete this thesis, I got to practise it a lot, and I feel like I definitely got better at that. Looking back, one thing I would do differently is the description of related work. At the start of the thesis, I put a significant amount of time into analysing related work. During this analysis, I took notes for every paper I read. I feel like, instead of taking the notes, I had better described interesting related work sources directly in my thesis after reading it. It would have saved me many hours since I had to revisit most sources when I started describing them in my thesis. Contrarily, I am glad that I started on time with the evaluation phase because although I knew a significant amount of time was necessary for this, it still took longer than I expected. Finally, I learned a lot about VPNs, fingerprinting techniques, and cyber security in general.

[2] https://github.com/1tayH/noisy (accessed on 20-05-2022)

Appendices

Appendix A

Dutch Summary

A.1 Inleiding

Er zijn vandaag de dag nog een aanzienlijk deel problemen omtrent de privacy van de gebruiker in de context van het afluisteren van het internetverkeer van de gebruiker. Wanneer een gebruiker met een bepaald eindpunt communiceert via het internet, zijn er een aantal instanties waarlangs het gegeneerde internetverkeer vloeit. Deze instanties hebben dan ook de mogelijkheid om dit voorbijkomende netwerkverkeer te inspecteren. In het algemeen bestaan er zo twee types instanties. Enerzijds is er de man in the middle (MITM) die zich initieel niet in het communicatiepad van de gebruik bevindt, maar zichzelf deze positie verschaft door bepaalde technieken uit te voeren. Dit type is vaak terug te vinden bij publieke hotspots aangezien eender wie kan connecteren met een hotspot. Dit toont ook aan dat publieke hotspots niet veilig zijn voor de privacy van de gebruiker [47]. Anderzijds zijn er instanties die zich standaard in het communicatiepad bevinden, zoals een internet provider.

Het is dus duidelijk dat het internetverkeer van een gebruiker beschermd moet worden tegen mogelijke afluisteraars. Een eerste poging om dit te verwezenlijken is de introductie van het HTTPS protocol waarbij de communicatie met de web server geëncrypteerd wordt en niet gedecrypteerd kan worden door afluisteraars. Op die manier kan gevoelige informatie, zoals bijvoorbeeld wachtwoorden, veiliger over het internet gestuurd worden. Indien enkel HTTPS gebruikt wordt, blijven er echter nog privacy problemen bestaan: afluisteraars kunnen nog altijd afleiden welke domeinen de gebruiker bezoekt door in de communicatie de IP adressen, DNS pakketten of TLS pakketten te analyseren. Om dit ook tegen te gaan, kan men gebruik maken van Virtual Private Networks (VPNs) of The Onion Router (Tor) [19]. VPN gebruikers sturen al hun netwerkverkeer naar een tussenliggende VPN server via een geëncrypteerde connectie. De VPN server stuurt dit netwerkverkeer vervolgens door naar de eigenlijke bestemming. Doordat het netwerkverkeer langs de tussenliggende VPN server loopt, wordt het IP adres van de eindbestemming verborgen voor afluisteraars en weet de eindbestemming het IP adres van de gebruiker ook niet. Dit biedt duidelijk een extra laag bescherming en privacy voor de gebruiker. Ons doel is om te onderzoeken of een afluisteraar toch nog informatie kan vergaren over een VPN gebruiker. Specifiek onderzoeken we of een afluisteraar de volgende elementen kan afleiden: besturingssysteem, web browser, type netwerkverkeer. We zullen twee verschillende methodes toepassen om dit te onderzoeken. In de eerste methode, de 'manuele methode' genoemd, analyseren we zelf het netwerkverkeer en proberen we features te ontdekken om te gebruiken voor classificatie van de verschillende aangehaalde types. Bij de andere methode, de 'machine learning methode' genoemd, zullen we machine learning (ML) classificeerders gebruiken voor classificatie. We stellen de volgende onderzoeksvragen op:

- Is het mogelijk om karakteristieken over het toestel van de VPN gebruiker af te leiden gegeven het netwerkverkeer dat uitgewisseld wordt tussen een VPN gebruiker en een VPN server?
  - Is dit mogelijk door een manuele methode te hanteren?
  - Is dit mogelijk door een machine learning methode te hanteren?
  - Hoe presteert een manuele methode ten opzichte van een machine learning methode?
- Wat kunnen VPN gebruikers doen om zich te beschermen tegen de toegepaste fingerprinting technieken?

A.2 Achtergrond

In dit hoofdstuk behandelen we een aantal belangrijke aspecten die aan bod komen in het verdere verloop van de thesis. Het eerste topic dat besproken wordt is cryptografie. Cryptografie is een term voor technieken die gebruikt worden om veilige communicatie te voorzien tussen twee eindpunten. Cryptografie verzorgt de vertrouwelijkheid en integriteit van de communicatie; dit zijn twee elementen van de 'CIA triad' [23], een model dat aan de basis ligt van het creëren van beveiligingssystemen. Er zijn twee soorten cryptografie: symmetrische en asymmetrische. Bij symmetrische cryptografie gebruiken beide eindpunten dezelfde sleutel voor encryptie en decryptie, bij asymmetrische gebruiken beide eindpunten een verschillende sleutel.

Een VPN biedt toegang tot een privénetwerk via een gedeeld netwerk. Dit wordt verwezenlijkt door het netwerkverkeer van de VPN gebruiker te tunnelen over het gedeelde netwerk. De term tunnelen refereert naar het encrypteren en encapsuleren van een netwerkpakket in een ander netwerkpakket. Op die manier kan netwerkverkeer dat bestemd is voor het privénetwerk over het gedeelde netwerk gestuurd worden. Een VPN is vaak noodzakelijk voor telewerken hetgeen hoofdzakelijk door de recente COVID-19 pandemie steeds meer toegepast wordt [50]. Sinds de start van de pandemie zien we in Europa daarom ook een toename in VPN gebruik [24]. VPNs worden echter ook gebruikt voor andere doeleinden waarvoor een VPN initieel niet bedoeld was. Zo is het gebruik van een VPN voor privacy/anonimiteit populair [64]. Het tunnelen zorgt er bijvoorbeeld voor dat een afluisteraar niet kan zien wat de gebruiker precies doet (privacy). Verder kunnen de eindpunten waarmee de VPN gebruiker communiceert het IP adres van de VPN gebruiker niet zien (anonimiteit). Een VPN introduceert echter ook nieuwe problemen omtrent privacy/anonimiteit omdat de VPN provider wel het netwerkverkeer en het IP adres van de gebruiker kan inspecteren. Uit een studie blijkt dat het VPN ecosysteem niet altijd even transparant is over in hoeverre dit toegepast wordt [38]. VPN protocols zijn verantwoordelijk voor het opzetten van de VPN tunnel. Er bestaan verschillende VPN protocols, de meest voorkomende zijn PPTP [78], IKEv2 [22], L2TP [72] en OpenVPN [52]. In deze thesis focussen we enkel op het OpenVPN protocol.

OpenVPN maakt gebruikt van een virtuele netwerkadapter, geïmplementeerd in software, die het netwerkverkeer van het toestel opvangt en dit doorstuurt naar OpenVPN software. De OpenVPN software vormt deze netwerkpakketten om tot tunnelpakketten die over het gedeelde netwerk gestuurd kunnen worden. De encryptie die toegepast wordt tijdens het tunnelen gebeurt met symmetrische cryptografie. Een belangrijk element dat we dienen te onthouden is dat enkel het volledige initiële gegenereerde pakket, dat getunneld wordt, geëncrypteerd is en gebruikt wordt als payload voor het tunnelpakket. De headers die het tunnelpakket vervolledigen tot een volwaardig netwerkpakket zijn niet geëncrypteerd. Verder beschikt OpenVPN over twee soorten kanalen: het controlekanaal en datakanaal. Het controlekanaal verzorgt de uitwisseling van parameters tussen client en server om een connectie op te kunnen zetten. Het datakanaal zorgt voor de eigenlijke uitwisseling van getunneld netwerkverkeer tussen de twee eindpunten, gebruik makende van de parameters die verkregen worden via het controlekanaal. Er zijn twee

protocols in de transportlaag die gebruikt kunnen worden voor de tunnel: TCP en UDP. TCP is de betere optie om firewalls te passeren terwijl UDP beter is voor performantie [15]. De onbetrouwbaardheid van UDP is geen probleem voor VPN tunnels omdat getunnelde TCP connecties dit oplossen. Het OpenVPN team raadt zelf ook aan om eerst UDP te proberen en bij problemen over te schakelen naar TCP [51]. Daarom focussen wij ons op UDP tunnels in deze thesis.

A.3 Fingerprinten van onderschept netwerkverkeer

Dit hoofdstuk bespreekt gerelateerd werk dat het fingerprinten van onderschept netwerkverkeer behandelt. Fingerprinten is het toepassen van patroonherkenning op netwerkverkeer om meer te weten te komen over de eindpunten die dat verkeer genereren. Fingerprinten is in de meeste gevallen een classificatieproces waarin beslist dient te worden welke classificatieoptie uit een lijst van mogelijkheden het meest waarschijnlijk is voor het geobserveerde netwerkverkeer. Fingerprinten kan op verschillende manieren toegepast worden, het is bijvoorbeeld mogelijk dat er informatie over een eindpunt afgeleid kan worden door gewoonweg ongeëncrypteerde elementen van het netwerkverkeer te inspecteren, hetgeen relatief eenvoudig is. Anderzijds kan men ook een stuk complexer te werk gaan door bijvoorbeeld bepaalde statistische waarden af te leiden en een ML algoritme te gebruiken voor patroonherkenning. Welke manier van fingerprinten gebruikt kan worden is deels afhankelijk van welke privacyvriendelijke technologieën aan bod komen in het netwerkverkeer. We beschouwen fingerprinten van netwerkverkeer waarbij minstens één van de volgende privacyvriendelijke technologieën gebruikt wordt: HTTPS, VPN of Tor.

We bespreken eerst fingerprinting methodes die enkel gebruik maken van ongeëncrypteerde elementen voor classificatie. Bij het gebruik van HTTPS kan een afluisteraar het domein bepalen dat een gebruiker bezoekt. Dit is mogelijk door rechtstreeks het domein af te lezen via DNS of TLS Client Hello netwerkpakketten. Indien deze pakketten geëncrypteerd zouden zijn [32,59], is dit echter nog altijd mogelijk op een indirecte manier door onder andere de ongeëncrypteerde IP adressen van de eindbestemmingen te gebruiken [18]. Er zijn verschillende methodes ontwikkeld om classificatie van besturingssystemen toe te passen door headers van netwerkpakketten te analyseren [4, 7, 13]. Hierbij wordt vooral naar TCP en IP headers gekeken. JA3 [6] maakt een hash van ongeëncrypteerde elementen in TLS onderhandelingspakketten. Deze hash dient als identificatie voor het eindpunt dat de desbetreffende pakketten genereerde. Op die manier kunnen bijvoorbeeld malware toepassingen gedetecteerd worden aangezien de communicatie tussen de client en server van dergelijke toepassingen typisch altijd exact hetzelfde verloopt, waardoor telkens dus dezelfde hash bekomen wordt. Daarnaast kan dit ook gebruikt worden om een bepaald type web browser met standaard configuratie te identificeren.
We zouden verwachten van VPNs dat er geen ongeëncrypteerde elementen aanwezig zijn die kunnen leiden tot classificatie van een bepaalde karakteristiek. Toch zien we vaak dat door fouten in de implementatie dat bepaalde netwerkpakketten niet langs de VPN tunnel worden gestuurd en dus niet afgeschermd worden door de VPN; dit is meestal het geval voor DNS en IPv6 [29, 35, 54, 74].

Er is een aanzienlijk deel gerelateerd werk dat fingerprinting methodes gebruikt waarbij er niet naar ongeëncrypteerde elementen gezocht wordt, meestal in de veronderstelling dat deze elementen niet aanwezig zijn. Dit is typisch het geval bij technologieën zoals VPN en Tor. Aangezien de manier waarop de classificatie bepaald kan worden minder triviaal is in dit geval, wordt vaak een ML algoritme gebruikt om patroonherkenning te gaan toepassen en zo te proberen een onderscheid te maken tussen de verschillende classificatieopties. De besproken werken worden samengevat en vergeleken in tabel A.1. De tabel geeft voor elke bron aan welk soort verkeer er wordt beschouwd (VPN of Tor), welk type classificatie er toegepast wordt, met welk soort ML algoritme dit gebeurt en ten slotte welke eigenschappen van het onderschept

Source	Traffic	Features	Classification	ML algorithm
Gerard-Gil et al. (2016)	VPN	Timing	Application type	C4.5 Decision Tree K-Nearest Neighbors
Gerard-Gil et al. (2016)	Tor	Timing	Application type	C4.5 Decision Tree K-Nearest Neighbors Zero R Random Forest
Shapira et al. (2019)	VPN	Timing Size	Application type	LeNet-5
Rimmer et al. (2018)	Tor	Raw trace	Webpage	Stacked Denoising Autoencoder Convolutional Neural Network Long-Short Term Memory
Bhat et al. (2019)	Tor	Raw trace Timing Direction Cumulative statistical features	Webpage	Var-CNN (ResNet-18 architecture)

Table A.1: Vergelijking van gerelateerd werk omtrent het fingerprinten van VPN en Tor netwerkverkeer

Web browser	Operating system	Traffic type	VPN
Google Chrome Microsoft Edge Mozilla Firefox	Windows 10 Ubuntu 20.0 LTS	Browsing YouTube (on-demand) Twitch (live)	OpenVPN (UDP)

Table A.2: Overzicht van gekozen parameters voor de manuele methode

netwerkverkeer gebruikt worden.

A.4 Fingerprinten van besturingssysteem, web browser en verkeer: manuele methode

In dit hoofdstuk bespreken we de creatie van een manuele methode om de volgende karakteristieken van een VPN gebruiker te fingerprinten: type besturingssysteem, type web browser en type netwerkverkeer. De gekozen classificatie opties en VPN worden weergegeven in tabel A.2. We starten door de volgende veronderstellingen te maken:

- Een VPN gebruiker is geconnecteerd met een VPN server en tunnelt zijn netwerkverkeer, dat hoofdzakelijk gegenereerd wordt door een desktop browser, door de VPN connectie.
- De VPN gebruiker gebruikt maar één type web browser tegelijk.
- De VPN gebruiker genereert geen aanzienlijke proportie netwerkverkeer in de achtergrond.
- Een aanvaller bevindt zich in het communicatiepad tussen de VPN gebruiker en de VPN server en kan deze geëncrypteerde communicatie dus onderscheppen en bekijken.
- De aanvaller kan de onderschepte, geëncrypteerde communicatie enkel observeren, betekenende dat hij deze bijvoorbeeld niet kan decrypteren.

Vervolgens voeren we een kleinschalig experiment uit waarin we traces genereren voor alle mogelijke combinaties van classificatieopties van verschillende types (bv. Windows-Chrome-Browsing). We visualiseren elke trace door deze voor te stellen als een histogram waarbij de x-as de tijd bevat en de y-as de grootte van het netwerkpakket. Deze visualisatiemethode hebben we overgenomen van de 'FlowPic' approach [65, 66] besproken in gerelateerd werk en vinden we zeer interessant om met het blote oog verschillen te kunnen observeren tussen verschillende traces. We zien duidelijke verschillen tussen de types verkeer, subtiele verschillen tussen de types web browser

Feature	Classifcation
TCP ACK	Traffic, OS
QUIC ACK	Traffic
TLS Client Hello	Traffic, OS
Packet rate	Traffic
IP TTL	OS
Max. packet length	Web browser

Table A.3: Overzicht van afgeleide types netwerkpakketten en de classificatie(s) waarvoor ze gebruikt worden

Classification	Accuracy
OS - approach A - 'unknown' excluded	97.36 %
OS - approach A - 'unknown' included	54.70 %
OS - approach B	100 %
Web browser	87.96 %
Traffic type	90.52 %

Table A.4: Nauwkeurigheid voor ieder type classificatie

en weinig tot geen verschil tussen de types besturingssysteem. Tijdens dit experiment identificeren we reeds een aantal mogelijkheden die we verder willen analyseren om later eventueel te gebruiken voor classificatie.

We stellen vast dat de lengte van het geëncapsuleerde pakket gevonden kan worden via het tunnelpakket. Dit komt omdat de standaard OpenVPN configuratie momenteel een encryptiealgoritme gebruikt dat de lengte van de inhoud niet verandert na encryptie. Hierdoor kunnen we, gegeven het tunnelpakket, de lengte van het geëncapsuleerde pakket bepalen. Deze lengte bekomen we door 52 bytes af te trekken van de totale lengte van het tunnelpakket van een IPv4 tunnel; voor een IPv6 tunnel bedraagt dit 72 bytes. Met behulp van deze observatie leiden we een aantal types geëncapsuleerde netwerkpakketten af uit tunnelpakketten die we kunnen gebruiken voor classificatie. In tabel A.3 tonen we de types afgeleide pakketten en de classificatie(s) waarvoor ze gebruikt worden. Tijdens dit proces van features ontdekken, hebben we besloten dat het moeilijk blijkt om een onderscheid te maken tussen de web browsers Chrome en Edge. We vermoeden dat de oorzaak is dat beide browsers Chromium als basis gebruiken en daarom gelijkaardig netwerkverkeer genereren. Daarom besluiten we om deze te combineren tot één optie, namelijk 'Chromium gebaseerd'. Verder hebben we ook twee verschillende methodes vastgesteld waarmee we het besturingssysteem kunnen classificeren, we noemen deze 'methode A' en 'methode B'. We zullen beide methodes evalueren. Methode A is een speciaal geval omdat er een extra classificatieoptie is toegevoegd, namelijk 'onbekend', aangezien het niet zeker is dat deze methode een resultaat zal teruggeven na een bepaald deel netwerkverkeer geanalyseerd te hebben.

We genereren zelf een dataset voor evaluatie van de manuele methode omdat geen enkele bestaande dataset hiervoor geschikt is. De dataset bestaat uit 1080 traces van elk 1 minuut. Voor elke trace in de dataset doen we telkens een predictie voor alle classificatietypes. De beschouwde statistieken voor evaluatie zijn nauwkeurigheid, precisie, sensitiviteit en F1-score. De formele definities voor deze statistieken zijn terug te vinden in sectie 4.5.2. De resultaten zijn weergegeven in tabel A.4 en A.5. Om de manuele methode live uit te testen, hebben we ook een tool geïmplementeerd die toelaat om live predicties te doen op basis van (live) geobserveerd OpenVPN netwerkverkeer.

Om zich te beschermen tegen de toegepaste fingerprintingtechieken, kunnen gebruikers een aan-

		Precision	Recall	F1-score
OS - approach A - 'unknown' excluded	**Windows**	100	94	97
	Linux	96	100	98
OS - approach A - 'unknown' included	**Windows**	100	47	64
	Linux	96	63	76
OS - approach B	**Windows**	100	100	100
	Linux	100	100	100
Browser	**Chromium based**	85	100	92
	Firefox	100	63	77
Traffic	**Browsing**	94	75	83
	YouTube	98	96	97
	Twitch	82	98	89

Table A.5: Precisie, sensitiviteit en F1-score voor alle classificatieopties

tal zaken ondernemen. Een eerste mogelijkheid is het toevoegen van padding waardoor de lengtes van de netwerkpakketten zullen veranderen. Daarnaast is het genereren van een aanzienlijke hoeveelheid achtergrondverkeer ook een optie. Verder kan men ook classificatieopties gebruiken die niet beschouwd zijn (bv. Safari web browser) en een andere privacyvriendelijke technologie gebruiken zoals Tor.

A.5 Fingerprinten van besturingssysteem, web browser en verkeer: machine learning methode

In dit hoofdstuk bespreken we de ML methode. Hierbij gebruiken we ML classificeerders uit gerelateerd werk in combinatie met de dataset die we genereerden voor de manuele methode. Specifiek kiezen we de 'FlowPic' [65,66] en 'Var-CNN' [10] classificeerders. We berekenen dezelfde statistieken als voor de manuele methode en vergelijken de resultaten verder ook met de manuele methode. De resultaten worden weergegeven in tabel A.6 en A.7. Merk op dat in beide tabellen methode B gekozen is voor het besturingssysteem van de manuele methode.

Classification	FlowPic	Var-CNN	Manual
OS	90.28 %	70.75 %	100 %
Browser - 3 options	67.59 %	54.29 %	/
Browser - 2 options	80.42 %	66.43 %	87.96 %
Traffic type	97.69 %	84.29 %	90.52 %

Table A.6: Nauwkeurigheid voor ieder type classificatie per gebruikte methode

		FlowPic			Var-CNN			Manual		
		P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
OS	**Windows**	90	90	90	81	54	65	100	100	100
	Linux	90	91	91	65	88	75	100	100	100
Browser - 3 options	**Chrome**	58	75	65	52	64	57	/	/	/
	Edge	71	54	61	48	47	47	/	/	/
	Firefox	80	74	77	67	51	58	/	/	/
Browser - 2 options	**Chromium based**	77	88	82	78	46	58	85	100	92
	Firefox	85	72	78	62	87	72	100	63	77
Traffic	**Browsing**	97	96	97	79	86	82	94	75	83
	YouTube	99	100	99	92	97	94	98	96	97
	Twitch	97	97	97	82	70	75	82	98	89

Table A.7: Precisie, sensitiviteit en F1-score voor alle classificatieopties per gebruikte methode

A.6 Conclusie

Het is mogelijk om karakteristieken over het toestel van de VPN gebruiker af te leiden mits we een aantal assumpties maken die niet altijd even realistisch zijn. In het algemeen presteert zowel de manuele methode als de machine learning methode goed. We concluderen dat geen van beide methodes de andere overtreft. De manuele methode presteert beter voor type besturingssysteem en web browser met twee classificatieopties terwijl de machine learning methode beter presteert voor type netwerkverkeer en web browser met drie classificatieopties. Om zich te beschermen tegen de toegepaste fingerprintingtechieken, kunnen gebruikers de volgende zaken ondernemen: padding toevoegen, achtergrondverkeer genereren, andere opties binnen een bepaald type classificatie gebruiken die niet beschouwd zijn (bv. Safari web browser) en een andere privacyvriendelijke technologie gebruiken zoals Tor.

Bibliography

[1] Desktop browser market share worldwide. Accessed on 20-05-2022. URL: https://gs.statcounter.com/browser-market-share/desktop/worldwide.

[2] Desktop operating system market share worldwide. Accessed on 20-05-2022. URL: https://gs.statcounter.com/os-market-share/desktop/worldwide/.

[3] IP in IP Tunneling. RFC 1853, October 1995. URL: https://www.rfc-editor.org/info/rfc1853, doi:10.17487/RFC1853.

[4] Ahmet Aksoy, Sushil Louis, and Mehmet Hadi Gunes. Operating system fingerprinting via automated network traffic analysis. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2502–2509, 2017. doi:10.1109/CEC.2017.7969609.

[5] Taher Al-Shehari and Sami Zhioua. An empirical study of web browsers' resistance to traffic analysis and website fingerprinting attacks. *Cluster Computing*, 21(4):1917–1931, 2018.

[6] John Althouse. Tls fingerprinting with ja3 and ja3s, Jan 2019. Accessed on 20-05-2022. URL: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967.

[7] Blake Anderson and David McGrew. Os fingerprinting: New techniques and a study of information gain and obfuscation. In *2017 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2017. doi:10.1109/CNS.2017.8228647.

[8] T Alexandra Beauregard, Kelly A Basile, and Esther Canonico. Telework: Outcomes and facilitators for employees. 2019.

[9] Michal Beno. The advantages and disadvantages of e-working: An examination using an aldine analysis. *Emerging Science Journal*, 5:11–20, 04 2021. doi:10.28991/esj-2021-SPER-02.

[10] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-cnn: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019:292–310, 10 2019. doi:10.2478/popets-2019-0070.

[11] Broadband. Key internet statistics to know in 2022 (including mobile), 2022. Accessed on 20-05-2022. URL: https://www.broadbandsearch.net/blog/internet-statistics.

[12] Sapna Chaudhary, Prince Sachdeva, Abhijit Mondal, Sandip Chakraborty, and Mukulika Maity. Youtube over google's quic vs internet middleboxes: A tug of war between protocol sustainability and application qoe, 2022. URL: https://arxiv.org/abs/2203.11977, doi:10.48550/ARXIV.2203.11977.

[13] Yi-Chao Chen, Yong Liao, Mario Baldi, Sung-Ju Lee, and Lili Qiu. Os fingerprinting and tethering detection in mobile networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, page 173–180, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2663716.2663745.

[14] Irfaan Coonjah, Pierre Clarel Catherine, and K. M. S. Soyjaudah. Experimental performance comparison between tcp vs udp tunnel using openvpn. In *2015 International Conference on Computing, Communication and Security (ICCCS)*, pages 1–5, 2015. doi:10.1109/CCCS.2015.7374133.

[15] Irfaan Coonjah, Pierre Clarel Catherine, and K. M. S. Soyjaudah. An investigation of the tcp meltdown problem and proposing raptor codes as a novel to decrease tcp retransmissions in vpn systems. In Suresh Chandra Satapathy, Vikrant Bhateja, Radhakhrishna Somanah, Xin-She Yang, and Roman Senkerik, editors, *Information Systems Design and Intelligent Applications*, pages 337–347, Singapore, 2019. Springer Singapore.

[16] Samantha Cossick. If the internet shutdown for a day, what would happen?, Sep 2021. Accessed on 20-05-2022. URL: https://www.allconnect.com/blog/what-would-happen-if-internet-down-for-day.

[17] E.F. Crist and J.J. Keijser. *Mastering OpenVPN*. Community experience distilled. Packt Publishing, 2015. URL: https://books.google.be/books?id=5VUqjgEACAAJ.

[18] Mariano Di Martino, Peter Quax, Wim Lamotte, and Neetesh Saxena. Knocking on ips: Identifying https websites for zero-rated traffic. *Sec. and Commun. Netw.*, 2020, jan 2020. doi:10.1155/2020/7285786.

[19] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[20] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001-11-26 2001. doi:https://doi.org/10.6028/NIST.FIPS.197.

[21] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346, 2012. doi:10.1109/SP.2012.28.

[22] Pasi Eronen, Yoav Nir, Paul E. Hoffman, and Charlie Kaufman. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996, September 2010. URL: https://rfc-editor.org/rfc/rfc5996.txt, doi:10.17487/RFC5996.

[23] Kristelle Feghali. The cia triad: Confidentiality, integrity and availability, Oct 2021. Accessed on 20-05-2022. URL: https://medium.com/coinmonks/the-cia-triad-confidentiality-integrity-and-availability-ef54d777cbd2.

[24] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. A year in lockdown: How the waves of covid-19 impact internet traffic. *Commun. ACM*, 64(7):101–108, jun 2021. doi:10.1145/3465212.

[25] Sheila Frankel, K. Robert Glenn, and Scott G. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 3602, September 2003. URL: https://www.rfc-editor.org/info/rfc3602, doi:10.17487/RFC3602.

[26] Sergey Frolov and Eric Wustrow. The use of tls in censorship circumvention. In *NDSS*, 2019.

[27] José Luis García-Dorado, Javier Ramos, Miguel Rodríguez, and Javier Aracil. Dns weighted footprints for web browsing analytics. *Journal of Network and Computer Applications*, 111:35–48, 2018. URL: https://www.sciencedirect.com/science/article/pii/S1084804518300894, doi:https://doi.org/10.1016/j.jnca.2018.03.008.

[28] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 world wide web conference*, pages 309–318, 2018.

[29] Fernando Gont. Layer 3 Virtual Private Network (VPN) Tunnel Traffic Leakages in Dual-Stack Hosts/Networks. RFC 7359, August 2014. URL: https://rfc-editor.org/rfc/rfc7359.txt, doi:10.17487/RFC7359.

[30] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. Characterization of encrypted and vpn traffic using time-related features. 02 2016. doi:10.5220/0005740704070414.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL: https://arxiv.org/abs/1512.03385, doi:10.48550/ARXIV.1512.03385.

[32] Paul E. Hoffman and Patrick McManus. DNS Queries over HTTPS (DoH). RFC 8484, October 2018. URL: https://www.rfc-editor.org/info/rfc8484, doi:10.17487/RFC8484.

[33] Matthias Horst, Martin Grothe, Tibor Jager, and Jörg Schwenk. Breaking pptp vpns via radius encryption. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security*, pages 159–175, Cham, 2016. Springer International Publishing.

[34] IANA. Service name and transport protocol port number registry. Accessed on 20-05-2022. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=19.

[35] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. An analysis of the privacy and security risks of android vpn permission-enabled apps. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, page 349–364, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2987443.2987471.

[36] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1143–1161. IEEE, 2021.

[37] Joshua Jones, Hayden Wimmer, and Rami J. Haddad. Pptp vpn: An analysis of the effects of a ddos attack. In *2019 SoutheastCon*, pages 1–6, 2019. doi:10.1109/SoutheastCon42311.2019.9020514.

[38] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M. Voelker, Alex C. Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez. An empirical analysis of the commercial vpn ecosystem. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, page 443–456, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3278532.3278570.

[39] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M. Voelker, Alex C. Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez. An empirical analysis of the commercial vpn ecosystem. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, page 443–456, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3278532.3278570.

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL: https://arxiv.org/abs/1412.6980, doi:10.48550/ARXIV.1412.6980.

[41] Michael K Kissi and Michael Asante. Penetration testing of ieee 802.11 encryption protocols using kali linux hacking tools. *International Journal of Computer Applications*, 176(32):26–33, 2020.

[42] Amit Klein and Benny Pinkas. Dns cache-based user tracking. In *NDSS*, 2019.

[43] Adam Langley. A Transport Layer Security (TLS) ClientHello Padding Extension. RFC 7685, October 2015. URL: `https://www.rfc-editor.org/info/rfc7685`, `doi:10.17487/RFC7685`.

[44] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 183–196, New York, NY, USA, 2017. Association for Computing Machinery. `doi:10.1145/3098822.3098842`.

[45] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)*, 14(2):1–33, 2020.

[46] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. `doi:10.1109/5.726791`.

[47] Ahmed Y. Lotfy, Alaa M. Zaki, Tarek Abd-El-Hafeez, and Tarek M. Mahmoud. Privacy issues of public wi-fi networks. In Aboul Ella Hassanien, Abdelkrim Haqiq, Peter J. Tonellato, Ladjel Bellatreche, Sam Goundar, Ahmad Taher Azar, Essaid Sabir, and Driss Bouzidi, editors, *Proceedings of the International Conference on Artificial Intelligence and Computer Vision (AICV2021)*, pages 656–665, Cham, 2021. Springer International Publishing.

[48] David McGrew and John Viega. The galois/counter mode of operation (gcm). *submission to NIST Modes of Operation Process*, 20:0278–0070, 2004.

[49] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[50] OECD. Teleworking in the covid-19 pandemic: Trends and prospects, Sep 2021. Accessed on 20-05-2022. URL: `https://www.oecd.org/coronavirus/policy-responses/teleworking-in-the-covid-19-pandemic-trends-and-prospects-72a416b6/#section-d1e858`.

[51] OpenVPN. Why does openvpn use udp and tcp?, May 2019. Accessed on 20-05-2022. URL: `https://openvpn.net/faq/why-does-openvpn-use-udp-and-tcp/`.

[52] OpenVPN, Nov 2021. Accessed on 20-05-2022. URL: `https://openvpn.net/`.

[53] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *WPES '11*, 2011.

[54] Vasile C. Perta, Marco V. Barbera, Gareth Tyson, Hamed Haddadi, and Alessandro Mei. A glance through the vpn looking glass: Ipv6 leakage and dns hijacking in commercial vpn clients. *Proceedings on Privacy Enhancing Technologies*, 2015(1):77–91, 2015. URL: `https://doi.org/10.1515/popets-2015-0006`, `doi:doi:10.1515/popets-2015-0006`.

[55] Vasile Claudiu Perta, M Barbera, Gareth Tyson, Hamed Haddadi, Alessandro Mei, et al. A glance through the vpn looking glass: Ipv6 leakage and dns hijacking in commercial vpn clients. 2015.

[56] Michael Peterson. Chapter 1 - maps and the internet: An introduction. In Michael Peterson, editor, *Maps and the Internet*, International Cartographic Association, pages 1–16. Elsevier Science, Oxford, 2003. URL: https://www.sciencedirect.com/science/article/pii/B9780080442013500037, doi:https://doi.org/10.1016/B978-008044201-3/50003-7.

[57] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, 2019. URL: https://doi.org/10.14722%2Fndss.2019.23386, doi:10.14722/ndss.2019.23386.

[58] E Ramadhani. Anonymity communication VPN and tor: a comparative study. *Journal of Physics: Conference Series*, 983:012060, mar 2018. doi:10.1088/1742-6596/983/1/012060.

[59] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-14, Internet Engineering Task Force, February 2022. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-14.

[60] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. 02 2018. doi:10.14722/ndss.2018.23115.

[61] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978. doi:10.1145/359340.359342.

[62] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288, August 2008. URL: https://www.rfc-editor.org/info/rfc5288, doi:10.17487/RFC5288.

[63] Bruce Schneier, Mudge, and David Wagner. Cryptanalysis of microsoft's pptp authentication extensions (ms-chapv2). In *Secure Networking — CQRE [Secure] ' 99*, pages 192–203, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[64] Security.org. Vpn consumer usage, adoption, and shopping study: 2021, Dec 2021. Accessed on 20-05-2022. URL: https://www.security.org/resources/vpn-consumer-report-annual/.

[65] Tal Shapira and Yuval Shavitt. Flowpic: Encrypted internet traffic classification is as easy as image recognition. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 680–687, 2019. doi:10.1109/INFCOMW.2019.8845315.

[66] Tal Shapira and Yuval Shavitt. Flowpic: A generic representation for encrypted traffic classification and applications identification. *IEEE Transactions on Network and Service Management*, 18(2):1218–1232, 2021. doi:10.1109/TNSM.2021.3071441.

[67] Yueshi Shen. Live video transmuxing/transcoding: Ffmpeg vs twitchtranscoder, part i. Accessed on 20-05-2022. URL: https://blog.twitch.tv/en/2017/10/10/live-video-transmuxing-transcoding-f-fmpeg-vs-twitch-transcoder-part-i-489c1c125f28/#:~:text=RTMP%20is%20a%20protocol%20designed,most%20video%20websites%20also%20use.

[68] Tanya Shreedhar, Rohit Panda, Sergey Podanev, and Vaibhav Bajpai. Evaluating quic performance over web, cloud storage and video workloads. *IEEE Transactions on Network and Service Management*, pages 1–1, 2021. doi:10.1109/TNSM.2021.3134562.

[69] Jinho Song, Yonggun Kim, and Yoojae Won. Operating system fingerprint recognition using icmp. In James J. Park, Doo-Soon Park, Young-Sik Jeong, and Yi Pan, editors, *Advances in Computer Science and Ubiquitous Computing*, pages 285–290, Singapore, 2020. Springer Singapore.

[70] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[71] Michael Tüxen, Fulvio Risso, Jasper Bongertz, Gerald Combs, Guy Harris, Eelco Chaudron, and Michael Richardson. PCAP Next Generation (pcapng) Capture File Format. Internet-Draft draft-tuexen-opsawg-pcapng-04, Internet Engineering Task Force, October 2021. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-tuexen-opsawg-pcapng-04.

[72] Andrew J. Valencia, Glen Zorn, William Palter, Gurdeep-Singh Pall, Mark Townsley, and Allan Rubens. Layer Two Tunneling Protocol "L2TP". RFC 2661, August 1999. URL: https://www.rfc-editor.org/info/rfc2661, doi:10.17487/RFC2661.

[73] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[74] Jack Wilson, David McLuskie, and Ethan Bayne. Investigation into the security and privacy of ios vpn applications. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3407023.3407029.

[75] Charles V Wright, Scott E Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, volume 9, 2009.

[76] Sami Zhioua. The web browser factor in traffic analysis attacks. *Security and Communication Networks*, 8(18):4227–4241, 2015.

[77] Sami Zhioua and Mahjoub Langar. Traffic analysis of web browsers. In *FMS@ Petri Nets*, pages 20–33. Citeseer, 2014.

[78] Glen Zorn, Gurdeep-Singh Pall, and Kory Hamzeh. Point-to-Point Tunneling Protocol (PPTP). RFC 2637, July 1999. URL: https://rfc-editor.org/rfc/rfc2637.txt, doi:10.17487/RFC2637.