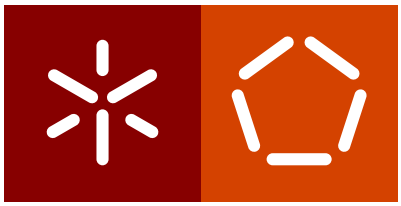
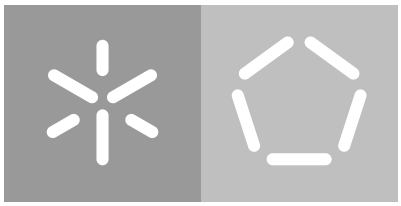


**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Bruno Rafael Lamas Corredoura Dantas

# Detection of Anonymized Traffic: Tor as Case Study

November 2019



**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Bruno Rafael Lamas Corredoura Dantas

# Detection of Anonymized Traffic: Tor as Case Study

Master dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
**Professor Maria Solange Pires Ferreira Rito Lima**
**Professor Paulo Manuel Martins Carvalho**

November 2019

## STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

# ACKNOWLEDGMENTS

I would like to express my thanks to some special people who were very important to me throughout this year and throughout my academic career.

Firstly, i want to thank my parents, Albertino and Olinda, and also my whole family, for always being present to me, and for the excellent education, comfort, and conditions they have given me.

To Professors Maria Solange, Paulo Carvalho and João Marco Silva, for all their availability and motivation they have given me throughout this year even when things did not seem to be going well, and for helping me overcome the problems that I faced during this work.

To my girlfriend, Raquel, for all the support and patience she had with me while I was doing this thesis.

To my close friends, who gave me (and give) the happiness to continue working and living every day with joy and enthusiasm.

To Rogério Costa, for his patience at work and his collaboration in reading and reviewing this document.

*Thanks to all of you!*
*Bruno Dantas*

## COPYRIGHT AND TERMS OF USE BY THIRD PARTIES

## ABSTRACT

This master thesis studies Tor, an anonymous overlay network used to browse the Internet. It is an open-source project that has gain popularity mainly because it does not hide its implementation. In this way, researchers and security experts can examine and confirm its security requirements.

Its ease of use has attracted all kinds of people, including ordinary citizens who want to avoid being profiled for targeted advertisements or circumvent censorship, corporations who do not want to reveal information to their competitors, and government intelligence agencies who need to do operations on the Internet without being noticed. In opposite, an anonymous system like this represents a good testbed for attackers, because their actions are naturally untraceable.

Traffic characteristics are studied in detail, which can be used to detect Tor. Further, a detection mechanism was developed to prevent users from reaching the Tor network. Finally, some changes are proposed so that Tor can better disguise its traffic with traditional web browsing traffic to overcome any intention of blocking it.



***Keywords***— anonymity, privacy, security, Tor, traffic classification

## RESUMO

Esta tese de mestrado estuda o Tor, uma rede *overlay* anónima que é usada para aceder a informações na Internet. Trata-se de um projeto *open-source* que ganhou popularidade principalmente porque não esconde a sua implementação. Desta forma, investigadores e especialistas em segurança podem examinar e confirmar os requisitos de segurança especificados.

A sua facilidade de uso atraiu diferentes tipos de pessoas, incluindo cidadãos comuns que pretendem evitar a publicidade e os anúncios direcionados ou a censura, empresas que não pretendem revelar informações aos concorrentes e agências de inteligência governamentais que precisam de realizar operações na Internet sem serem vigiadas. Por outro lado, um sistema anónimo como este representa um bom ambiente de teste para atacantes, porque as suas ações são difíceis de controlar.

As características de tráfego são estudadas em detalhe, e podem ser usadas para detectar o Tor. Além disso, foi desenvolvido um mecanismo de deteção para impedir que os utilizadores alcancem a rede Tor. Finalmente, são propostas algumas alterações para que o Tor possa ofuscar melhor o seu tráfego com o tráfego tradicional Web, de modo a ultrapassar qualquer intenção de bloqueá-lo.



***Keywords***— anonimato, classificação de tráfego, privacidade, segurança, Tor

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF LISTINGS

# ACRONYMS

**A**

**ACL** Access Control List.

**AEAD** Authenticated Encryption with Associated Data.

**AES** Advanced Encryption Standard.

**C**

**CA** Certificate Authority.

**D**

**DH** Diffie-Hellman.

**DNS** Domain Name System.

**DPI** Deep Packet Inspection.

**G**

**GCM** Galois/Counter Mode.

**GDPR** General Data Protection Regulation.

**H**

**HIDS** Host-based Intrusion Detection System.

**HTTP** Hypertext Transfer Protocol.

**HTTPS** Hypertext Transfer Protocol Secure.

**I**

**IANA** Internet Assigned Numbers Authority.

**IDS** Intrusion Detection System.

**IPS** Intrusion Prevention System.

**IPV4** Internet Protocol version 4.

**ISP** Internet Service Providers.

**IT** Information Technology.

**IV** Initialization Vector.

**L**

**LAN** Local Area Network.

**M**

**MAC** Message Authentication Code.

**N**

**NAT** Network Address Translation.

**NIDS** Network-based Intrusion Detection System.

**O**

**OSI** Open Systems Interconnection.

**P**

**PII** Personal Identifiable Information.

**PKI** Public Key Infrastructure.

**Q**

**QOE** Quality of Experience.

**QOS** Quality of Service.

**R**

**RFC** Request For Comments.

**RSA** RivestShamirAdleman.

**S**

**SNI** Server Name Indication.

**SSL** Secure Sockets Layer.

**T**

**TCP** Transmission Control Protocol.

**TE** Traffic Engineering.

**TLS** Transport Layer Security.

**TOR** The Onion Router.

**TPT** Tor Pluggable Transports.

**U**

**URL** Uniform Resource Locator.

**V**

**VOIP** Voice over Internet Protocol.

**VPN** Virtual Private Network.

# 1

# INTRODUCTION

## 1.1 MOTIVATION

Privacy is a human right and online privacy should be no different. While communications and data need firm protections online, bureaucracy has been slow to respond to the pace of technological change.

In a world with so many massive databases of personal information, the chance of a data breach goes through the roof. In Europe, even with the introduction of the *General Data Protection Regulation (GDPR)*[1], there is no guarantee that organizations comply with it [2]. By having more data about individuals, organizations can be more effective with their marketing and deliver highly targeted advertisements. For example, a 50 million euro fine was applied by the French data protection authority to Google, due to the firm processing personal data for advertising purposes without valid authorization.

The lack of trust in the *Information Technology (IT)* domain has led individuals to discover different ways of hiding their identities online (online anonymity). Despite the privacy invasion by companies, governments' agencies and censors also come into play. The main argument against online anonymity by governments is about users having a lack of accountability. In other words, anonymity can harbor criminal activity by making the tracing of online activities more difficult [1]. Just like governments want to control citizens, censors monitor online activity looking for users that violate established countries' rules. In more dictatorial countries, the Internet has already been turned off to obfuscate internal problems from reaching the rest of the world. Further, anonymous traffic hardens the management and monitoring of network infrastructures because the traffic cannot be easily associated to its original sources and/or destinations. In this context, detecting and blocking anonymous traffic may be important, in some cases, to the good operation of the network.

This work studies *The Onion Router (Tor)*, a tool that allows its users to achieve online anonymity. By using Tor, users can access the public Internet without worrying about censors, governments, service providers, and so on. The motivation of this work is to take Tor one step further in its continuous research and help it to make online privacy a reality.

1 The essentials of this regulation is to use as minimum *Personal Identifiable Information (PII)* as possible, and to guarantee that this data is properly managed and secured.

2 https://www.helpnetsecurity.com/2019/02/07/gdpr-numbers-january-2019/

## 1.2 OBJETIVES

The main goal of this work is to detect the presence of Tor in a network in order to prevent users within a network from using it, if required. The starting point consists of studying Tor. Understanding how it provides online anonymity and examining its characteristics may provide a means of choosing the right variables to detect in the blocking process. This blocking process can result from proper actions taken by, for instance, an *Intrusion Detection System (IDS)*.

Furthermore, this work studies if Tor is, or is not, detected by external observers. The detection of Tor means that more research must be done. In opposite, showing that Tor is undetectable ensures its usability, and consequently, its scalability.

## 1.3 RESEARCH QUESTIONS

- What are the characteristics of Tor traffic in terms of traffic analysis?
- How does Tor provide online anonymity and how is it different from other traffic?
- Are there any changes that could make Tor look like common traffic?
- Can Tor traffic be effectively detected?

## 1.4 CONTRIBUTIONS

First, several differences were discovered between Tor and other types of traffic. More specifically, between the Tor Browser and other browsers studied. Second, these differences were expressed as IDS signatures to detect traffic originated from Tor, proving that Tor can be detected. Finally, some tests were performed to test the detection accuracy by mixing Tor with other kinds of traffic.

## 1.5 THESIS LAYOUT

This thesis is organized as follows. Chapter 2 contains all the background information to understand the rest of the work. Chapter 3 contains the dissemination of Tor that is then used to create the proper signatures. Chapter 4 explains the tests performed to evaluate the detection of Tor and the signatures created, and ends with results' discussion. Finally, Chapter 5 contains some conclusions and addresses the future in this area of research.

# 2

# STATE-OF-THE-ART

This chapter presents concepts useful for understanding the rest of the work. Initially, a discussion about *traffic classification*, more specifically its evolution and why it has been giving special attention in the last few years. Then, *anonymity systems* that are used to prevent network tracking and minimize privacy issues deserve a general discussion (although this work focuses on Tor). Two sections are dedicated to explain the security protocol that Tor uses and Tor itself. The topic of the fifth section is IDS, security tools that are used to prevent malicious or suspected traffic from reaching a network or a single host. The chapter ends with a revision of Tor related works.

## 2.1 TRAFFIC CLASSIFICATION

The discipline of traffic classification tries to associate traffic flows or packets with the applications, or application types, that generated them. A traffic flow can be thought of as a single instance of an application-to-application flow of packets identified by some pre-defined parameters. A common one in the context of computer networking is the five tuple: source address, source port, protocol, destination address, and destination port. This tuple is used to (uniquely) identify different flows, and is used since the first-generation firewalls [2]. Back in the early days of the Internet, a small number of applications with known fixed assigned port numbers, added to the fact that security was not of major concern, has led to simple approaches for traffic classification. *Port-based* approaches - checking packets' port numbers - was enough to achieve high accuracy results [3]. However, over the last two decades, some developments make it difficult for operators and service providers to classify traffic flows:

- applications that have no *Internet Assigned Numbers Authority (IANA)* registered ports, but instead use ports already registered, randomly selected, or user-defined;
- the use of well-known ports to circumvent filtering or firewalls;
- *Internet Protocol version 4 (IPv4)* address exhaustion and *Network Address Translation (NAT)* usage, where several physical servers may offer services through the same public IP address but on different ports.

Operators were forced to use another approach, commonly called *Deep Packet Inspection (DPI)*[1], by looking at packets' content to discover the application being used [4]. This approach has two downsides. The use of pattern matching can become easily slow because each incoming packet has

1 To protect the users privacy, some legal restrictions may be imposed to prevent the access to the payload of the packets.

to be compared with thousands of different *signatures* [5]. Also, end-to-end encryption is becoming ubiquitous, which makes it practically useless [6]. Note, however, that this does not mean that the port-based approach is no longer used. Because checking packets' header is faster than checking the payload, this approach can still be a good option in distinguishing traffic at high granularity levels or in combination with other approaches.

### 2.1.1 *Encrypted data*

With the universal adoption of data encryption, motivated by security and privacy issues, classifying traffic became more difficult [6]. At first glance, with the previous considerations, it seems that none of the layers could be used to infer applications or application types. However, some information can be extracted from encrypted connections, typically from the sessions initiation.

The reason for this is that security protocols usually have an initialization phase that is not encrypted. This phase can be further divided into an initial handshake, an authentication phase, and a shared secret establishment. The responsibility of this phase is to prepare all the necessary parameters for (encrypted) data transfer.

With the increasing complexity of networks, the classification methods are usually supported with the help of protocols knowledge [6]. For example, the introduction of the *Server Name Indication (SNI)* extension in the *Transport Layer Security (TLS)* handshake (Client Hello message) process reveals the name of the server that the user is accessing in plaintext. As already stated, NAT usage by companies has some limitations. One of them occurs when a company has multiple different services under a smaller number of public IPv4 addresses. When the mappings between IPv4 addresses and services is not one-to-one, the server does not know the certificate to retrieve to the client, because the client only wants access to one of the services. The extension was introduced to prevent this, and when a server receives it from the client, it immediately retrieves the corresponding certificate. Figure 2.1 illustrates the two cases, with and without the extension. The knowledge of the previous example would let one create specific signatures to decide whether to filter access to specific services.

Note that encrypted traffic does not prevent well-known services, that have well-known IPv4 addresses, from being classified. The traffic can be encrypted, but it is possible to inspect the network layer information (IPv4 address) to associate that flow with the (well-known) service being used. In an attempt to avoid being related to a specific well-known service, users may use a proxy. With SNI, however, using a proxy may not solve the problem. Before a packet can reach the proxy, DPI techniques can be used to inspect this field in the TLS handshake and infer the service being used by the user. A possible solution would be to first establish a tunnel with the proxy and only then access the service. In that case, the SNI extension would be encrypted until it reached the proxy.

This previous discussion highlights the arms-race between traffic classification and traffic obfuscation. Because of that, in recent years, this area has been giving lots of attention, with trending of coupling traffic classification with other approaches, such as machine learning, deep learning, and specific heuristics [3][6]. These approaches are mainly applied to the specific *network metrics* of each application characteristics, at different granularity levels. This means that researchers are testing different metrics combinations to achieve high accuracy classifiers.

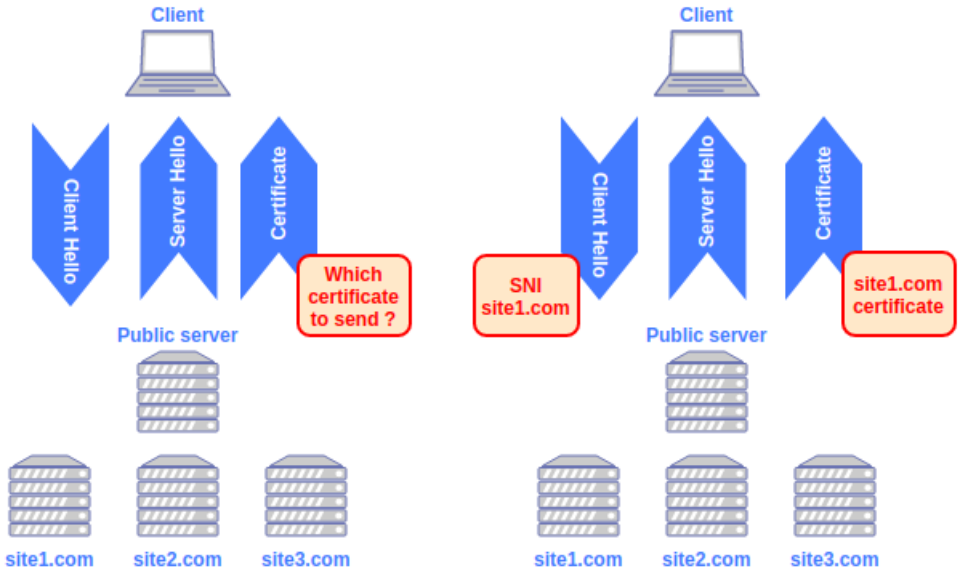



Figure 2.1: Server behavior (a) without SNI and (b) with SNI

Despite this, classifying encrypted traffic is not always possible, especially if operators' techniques are publicly known. In this case, developers intentionally change the application's network metrics (e.g. *Tor Pluggable Transports (TPT)*). By modifying them (e.g. packets' inter-arrival times, packets' length, using dynamic ports) it is possible to obfuscate the application generating the traffic. Obviously, it is easier to distinguish the application type rather than the applications itself. An application type has its own traffic characteristics and can be identified by simply eavesdropping the network connection. For example, the differences between *inelastic* and *elastic* applications can be more easily distinguished, even if packets are encrypted, because bandwidth requirements differ a lot between them. However, knowing that it is an inelastic application, distinguish between *Voice over Internet Protocol (VoIP)* or live video stream can become harder. Further, imagine that, with high accuracy, the traffic is known to be generated from a VoIP application. Discovering the application itself is even more difficult. To support this, in [7], the authors could increase the *Quality of Service (QoS)* by distinguishing VoIP traffic from other types of traffic, but they could not distinguish VoIP applications between themselves.

This emerging discipline is important because it can be used in areas such as QoS, *Quality of Experience (QoE)*, *Traffic Engineering (TE)* or *Cybersecurity* [4]. It can also be used to perform online tracking or targeted advertising. In any case, some control over the infrastructure will always be necessary for its proper functioning. Often, the only way to achieve this is by inspecting packets, and possibly personal information. For this reason, this issue generates some controversy.

## 2.2 Online Anonymity Systems

The main goal of *anonymity systems* is to avoid traffic analysis and network surveillance, and to block any tracking of users' identities in the Internet [8]. Why is it needed, if almost all applications' data is encrypted? First, service providers usually have logging systems to monitor their infrastruc-

ture. The recorded logs can have information that can be used to keep track of users' identities or activities. Secondly, secure communication protocols can reveal information or have implementation flaws. For example, the *Client Hello* message in the TLS handshake can carry the server name in plaintext (SNI extension). Third, as already said, traffic classification techniques are starting to adopt statistical approaches to classify encrypted traffic, based on network and transport layers information or communication metrics. In other words, it is still possible to guess, with some level of certainty, what kind of services users are accessing, even if the communications are encrypted.

These discussed points are the reasons why most of the anonymity systems use techniques in network or transport layers. In simple words, the goal is to separate the application data from the lower level layers, as a way to obfuscate who is accessing the service. Typically this is achieved by bouncing the traffic to an intermediate entity before accessing the final service. That entity then uses some strategy to obfuscate the traffic (these strategies are discussed through the next subsections).

On the one hand, consumers have the interest in hiding their identities by accessing anonymity systems. On the other hand, some entities want to monitor network traffic for different purposes and are against those tools [6]. If the linkage between a user and a service does not exist it does not matter what service is being accessed. As so, control entities can know that a user is using an anonymity system, not the service. Similarly, service owners cannot know the user accessing it, only that the requests came from an anonymity system.

Depending on the architecture, anonymity systems can be used to circumvent geographically blocked content, dodge targeted marketing or test network attacks. Also, they are a troublesome for governments in censorship countries, as they allow citizens to access censored websites without being discovered. As a consequence, those countries' government agencies and particular service providers block traffic generated or sent to those tools. When an anonymity system starts to become more famous, the simplest approach is to block their public IPv4 addresses [9]. In this way, citizens are forced to access the service without an anonymity system or to find another system less famous and not blocked yet. Without directly blocking anonymity systems, these agencies would have to apply other traffic classification techniques already discussed, which is more difficult. If the traffic is classified by belonging to some anonymity system and is blocked, citizens are unable to access whatever they like through that system (even if what they would like to access is legal).

Currently, there is no anonymity system capable of leaving all parties satisfied. These systems use techniques such as proxies, mix networks, tunneling and overlay networks. Each one has its own design, which means that each one has its specific use case, advantages and disadvantages. This thesis will focus on Tor, an anonymity system that uses an overlay network to provide anonymity to its users. Before discussing that system, the following sections present some common design choices about anonymity systems.

### 2.2.1 *Proxies*

In the simplest case, users connect to a single proxy, and this to the destination. You can think of this as a man-in-the-middle entity. There are two different approaches: Centrally-controlled shared proxies and Independent personal proxies. In the first (Figure 2.2), companies behind these tools can have more than one proxy, controlling and operating them centrally, with many different users getting assigned to only one proxy. In the latter (Figure 2.3), companies deploy applications that

users can install on their computers. Each user runs a proxy, and relays its own traffic to another proxy/user. The last approach is slightly better for anti-blocking purposes because it creates smaller clusters of traffic. If each proxy only has a few users, and there is no central list of proxies, most of them will never get noticed by censors or service providers.

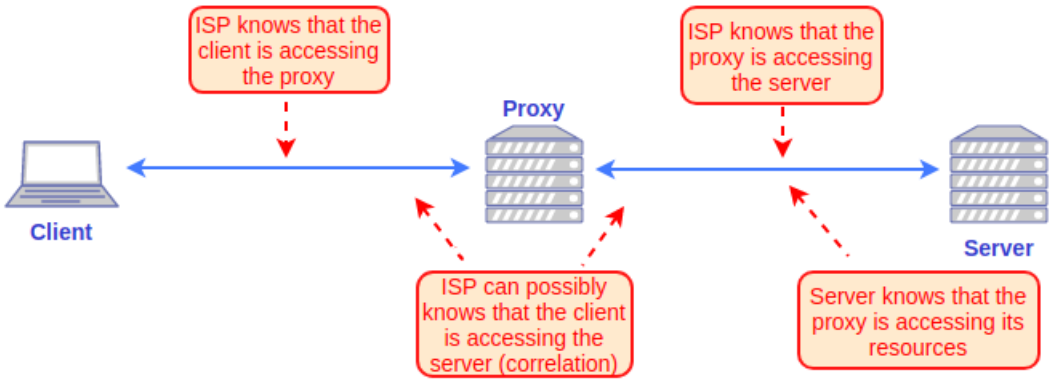



Figure 2.2: Centrally controlled proxy [9]

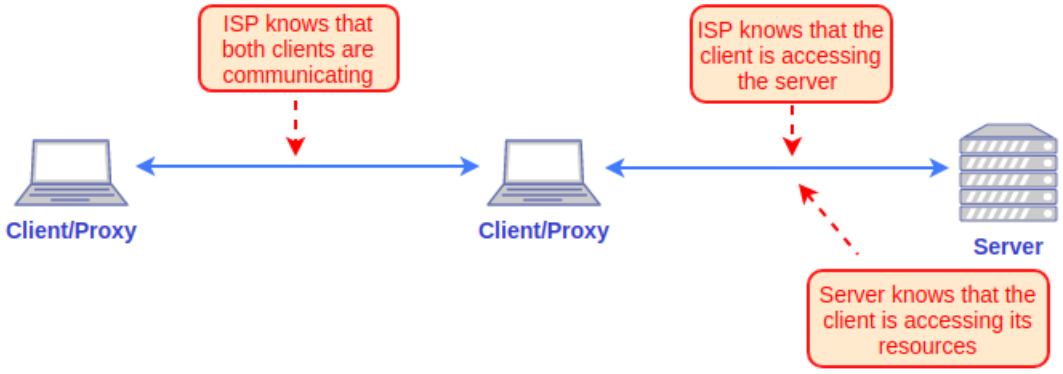



Figure 2.3: Independent personal proxies [9]

In general, the main advantage of single proxies is that traffic analysis becomes more difficult. Also, these systems have better performance than systems that bounce traffic through more than one relay.

Single proxies provide weak security compared to systems that distribute trust over multiple relays. First, a compromised proxy can trivially observe all of its users actions. Second, an eavesdropper only needs to monitor a single proxy to perform timing correlation attacks against all its users traffic and thus learn where everyone is connecting. Third, all users need to trust the proxy company to have good security itself as well as to not reveal user activities. Finally, even if a company has multiple proxies, a single user connects to just one, which presents a single point-of-failure.

Generally, when a company starts becoming popular, it is blocked by its IPv4 addresses[6]. Then, they start renting lots of disparate addresses and rotating through them as they get blocked. To circumvent this problem, they start sending the current addresses through other means (e.g. email) [9].

### 2.2.2 *Mix Networks*

Mix networks use a chain of proxy servers known as *mixes*. As the name suggests, each incoming message is mixed (shuffled) with several other messages. Outgoing messages are destined to another random proxy. This behavior can occur as many times as needed. Figure 2.4 represents a case where a client's messages traverse at least two proxies. Note that in a real scenario, more clients may be using the infrastructure, which reinforces the anonymity of the system (usability hardens correlation attacks).

As with single proxies, these systems break the link between the source of the request and the destination, making it harder for eavesdroppers to trace end-to-end communications. In the case where all mixes belong to the same entity, the problem of trusting that entity to be benign persists. If mixes are controlled by different entities, each one only knows the node that it immediately received the message from, and the immediate destination to send the shuffled messages to, making the network resistant to malicious mix nodes.

Note that now establishing a correlation is more difficult, but this architecture also increases the overall latency of the system.

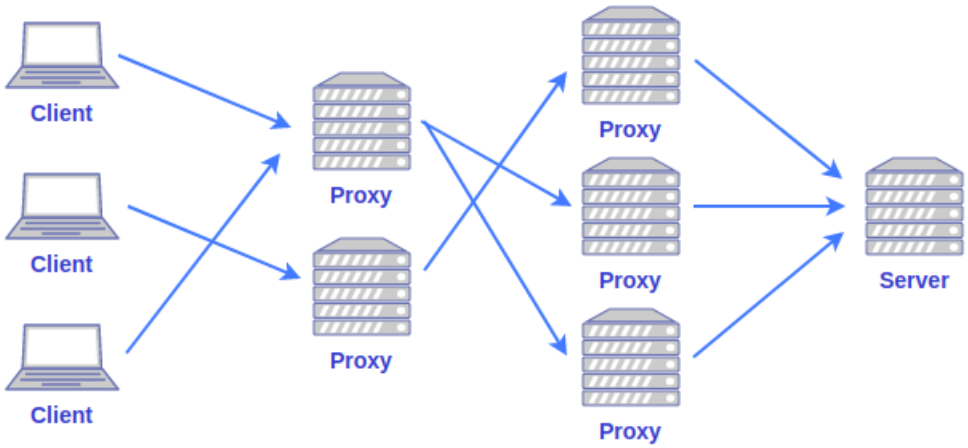



Figure 2.4: Mixing networks

### 2.2.3 *Tunnelling*

This technique, many times associated with *Virtual Private Network (VPN)*, establishes a secure tunnel between the client and a gateway. The latter then communicates with the service that the client wants to access. The difference to single proxies is that a tunnel hides and possibly encrypts all the traffic, not only web traffic. Because of that, all applications used by the client that require Internet access will be secured until reaching the gateway. However, because the discussion is only about anonymity, the same principles that apply to singles proxies, also apply here. This architecture is illustrated in Figure 2.5. The pipe is used to highlight that all traffic is hidden, and the intermediate node is the gateway.

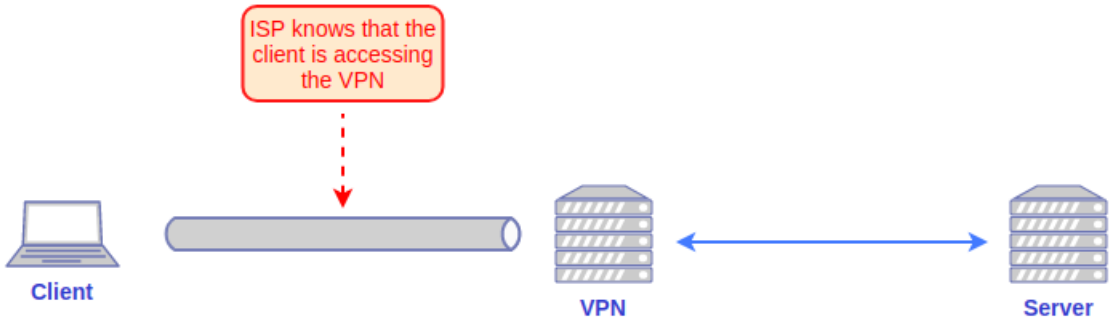



Figure 2.5: Tunneling

### 2.2.4 *Overlay networks*

The last architecture to discuss are overlay networks, which are nothing more than logical networks on top of the existing network. The best way to think of an overlay network is as a network that defines its own nodes and paths on upper layers. For instance, in the existing layer 3 networks, the path taken by packets is normally handled by devices with layer 3 capabilities (sometimes more). In an overlay network, the network is controlled by upper-layer devices. Note that with overlay networks, if layer 3 paths fluctuate, the logical path remains the same. Also, a complete path in the overlay network may require many paths in the physical network.

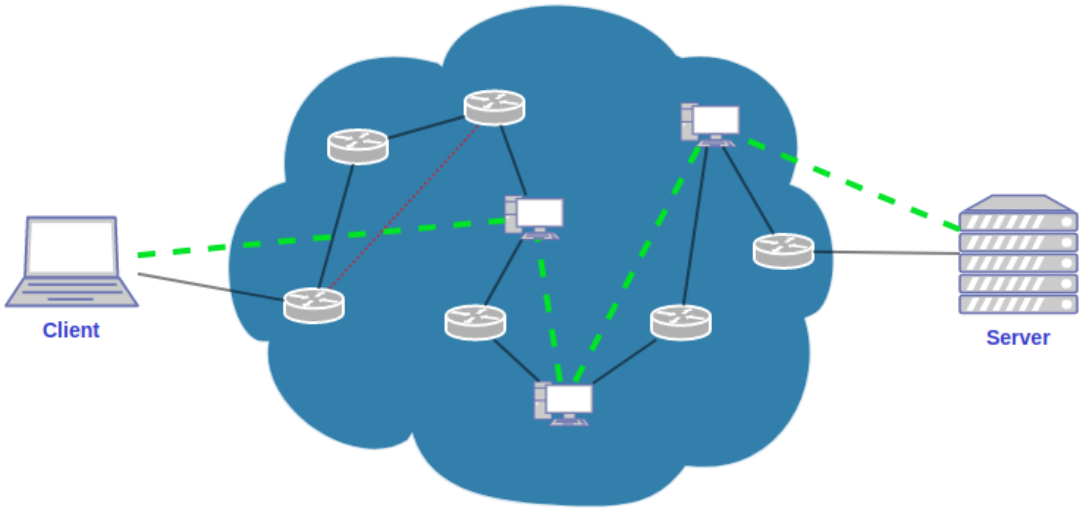



Figure 2.6: Overlay network

In Figure 2.6 the black lines represent physical links interconnecting routers, the green dashed lines represent the logical path in the overlay network, and the blue cloud represents the public internet. The red dotted line illustrates that a physical path can change, but the logical remains the same. Packets sent from client to server (and vice-versa) can traverse different routers along the way but must traverse each upper layer devices.

The difference to mix networks is that in an overlay network a path/circuit is built. In mix networks, the selection of the next proxy/node to send each message can vary. It is important to understand this architecture, because Tor relies on it to provide anonymity to its users.

## 2.3 TLS

TLS, as the name suggests, provides security above the transport layer in the *Transmission Control Protocol (TCP)*/IPv4 model. It has an intimate relationship with its predecessor, *Secure Sockets Layer (SSL)*. The latter was first introduced in 1994 by Netscape Communications, but that version was never published. In the next year, SSLv2 was published and did not last too long. Like the first version, they both had security flaws. Therefore, SSLv3 had a complete redesign and was published in 1996.

The TLS protocol was first introduced in 1999 as an upgrade to SSLv3. From this point, TLS has been improved twice. The actual version is TLS 1.3, which obsoletes previous versions. However, in this section, only TLS 1.2 is described because is the current version supported by Tor. There is already a change proposal in Tor's repository to migrate to TLS 1.3 [2]. From this point on, TLS 1.2 will be referred simply as TLS.

### 2.3.1 *Overview*

To better understand this protocol, notice that it lays between transport and application layers in the TCP/IPv4 model, or in the *Session Layer* in the *Open Systems Interconnection (OSI)* model. Figure 2.7 tries to illustrate where the two TLS layers reside in the TCP/IPv4 model (there are two logic layers with respect to TLS). At the edges, there are the two well-known layers, Transport and Application. In the middle, the lower layer is the *TLS Record Protocol*, which is used in all the messages sent/received. Basically, this protocol is responsible of preparing packets between the two adjacent layers (encrypt/decrypt, verify, compress/decompress, fragment/reassemble). The upper layer can be one of three *TLS Handshaking Protocols* or the TLS *Application Protocol*. The three Handshaking Protocols are: *Handshake Protocol*, *ChangeCipherSpec Protocol* and *Alert Protocol* [10].

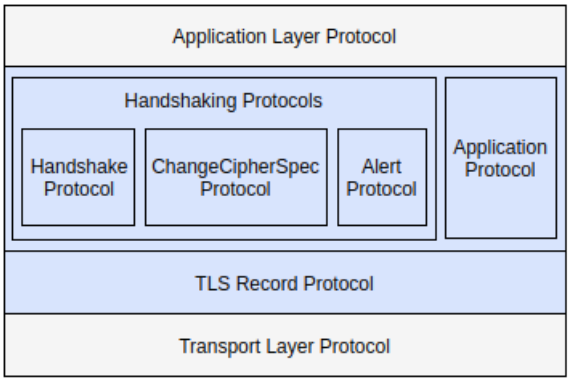



Figure 2.7: TLS protocol and its corresponding layers [10]

2 https://gitweb.torproject.org/torspec.git/tree/proposals/294-tls-1.3.txt

### 2.3.2 *Record Protocol*

Every message sent to or received by a host uses the TLS Record Protocol. This fact is also illustrated in Figure 2.7, which shows that the corresponding layer has only this protocol (messages need to traverse the layer because there is only one choice). In contrast, in the upper layer, messages can only use one of four different protocols.

A brief description from the client-side perspective is better to explain this protocol. For this, suppose that a client application, relying on TLS, generates a message to be transmitted to the server. First, this message is encapsulated in one of the four protocols mentioned above. Then, the Record Protocol takes that message, **fragments** it into blocks if necessary, **compresses** the data (optional), applies a *Message Authentication Code (MAC)* and finally **encrypts** it. When a response is sent by the server, the message (received from the transport layer) is **decrypted** (and **verified**), **decompressed**, **reassembled**, and delivered to one of the four protocols above it.

All of these operations are performed from a **connection state**, an operating environment of the TLS Record Protocol. This state begins with predefined (null) values. If this was not the case, the client and the server would not be able to understand each other. To actually fill the values of the connection state, the client and the server must agree on a set of security parameters, which is achieved with the Handshaking Protocol. After establishing these parameters, six items are generated from them: client write MAC key, client write encryption key, client write *Initialization Vector (IV)*, server write MAC key, server write encryption key, server write IV.

A subtle detail is that all the operations described above are only really applied when a ChangeCipherSpec message is sent. Basically, the connection state has a pending and a current state, and the Record Protocol only relies on the current. If a new connection is being established, both states (current and pending) will start with empty values. Performing the handshake will change the pending state, which is not used yet. Sending a ChangeCipherSpec message will force the current state to override its values with the pending state values. Only after this message, both parties are able to securely exchange messages with the agreed parameters.

After discussing how messages are treated, its time to discuss messages' format. Figure 2.8 shows the format of Record Protocol messages. The `Type` field is a one-byte value that indicates one of the four higher-level protocols used to process the message. The `Version` field indicates the current version being employed (version 1.2 in this work). By historical reasons, this version has the value {3, 3} (TLS 1.0 started as {3, 1}, a continuation of SSLv3, hence the values) which corresponds to two-bytes, each one with the value 3. `Length` indicates the length of this fragment in bytes, and the rest of the message is filled with the higher-level protocol.

### 2.3.3 *Handshaking Protocols*

The Handshaking Protocols sit above the Record Protocol, and is composed of three different protocols. Remember that already two of them have been mentioned, Handshake Protocol and ChangeCipherSpec Protocol. These are used to provide security parameters and to update the state to these parameters, respectively. The Alert Protocol is used to report warnings or fatal errors. Each protocol has a specific message format, discussed through the following subsections.

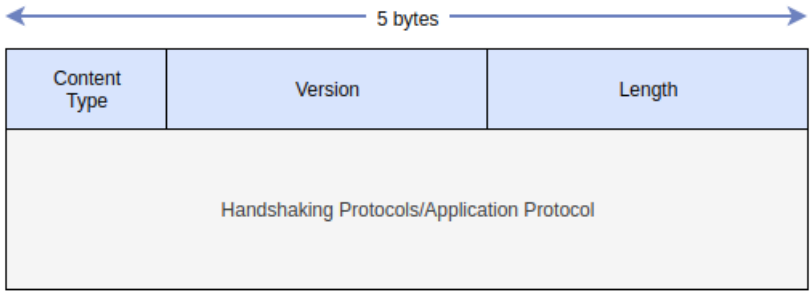



Figure 2.8: TLS Record Protocol message format [11]

*Handshake Protocol*

In this protocol, the first byte of the message (`Type`) defines the handshake message type (there are 10 different types). The following three bytes define the `Length` of the message. Note that the length space in this message (3 bytes) is greater than the length space in the Record Protocol (2 bytes). However, the latter can fragment messages passed down if it exceeds the 2-byte length space. The rest of the message will depend on the handshake message type (Figure 2.9).

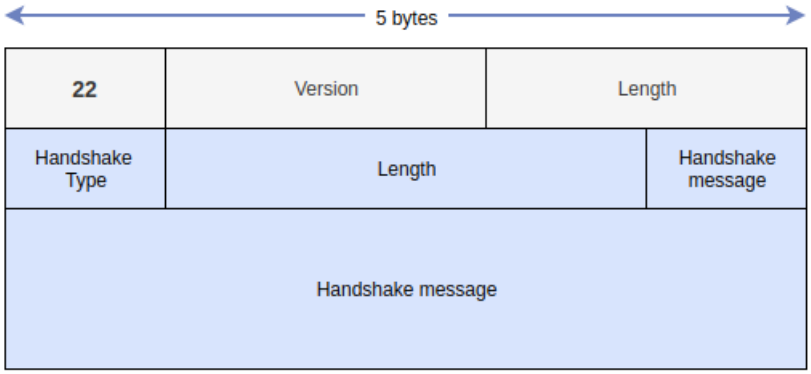



Figure 2.9: TLS Handshake Protocol message format [11]

The 10 different types of handshake messages are `Hello Request`, `Client Hello`, `Server Hello`, `Certificate Request`, `Certificate`, `Certificate Verify`, `Client Key Exchange`, `Server Key Exchange`, `Server Hello Done` and `Finished`. Because this protocol is the most exhaustive among the 4 protocols, Figure 2.10 illustrates a flowchart of the previous messages in a complete handshake with no session resumption.

The words in bold mean that messages are encrypted. The character * indicates that messages are optional. Typically, servers do not send the `Certificate Request` message to the client (which does not send his `Certificate` and `Certificate Verify` messages). Also, `ChangeCipherSpec` is embraced in square brackets to emphasize that this message is itself a protocol. Each handshake message type has another specific format, which is not covered here. However, a brief description of each message type and the most important fields follows:

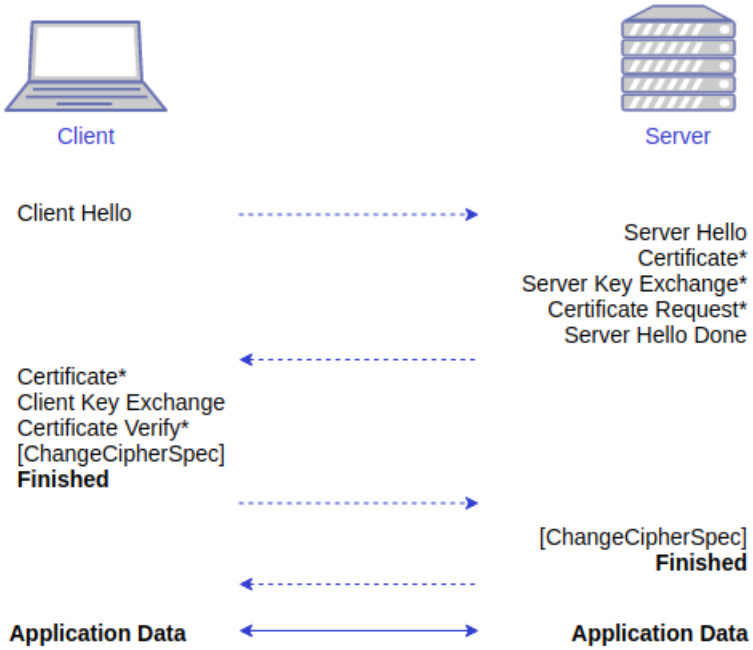



Figure 2.10: TLS Handshake Protocol flowchart [10]

- **Client Hello -** sent to initiate a connection with the server (or in response to a `Hello Request` from the server).
  - `Version` - specify the latest (high-valued) version that the client wish to use.
  - `Random structure` - the current time followed by a random generated number (used latter in the protocol to increase keys generation entropy).
  - `Session ID` - is the current session identifier. It is often used to decide whether a connection needs to be established from scratch (the server does not know this identifier) or resumed from an old connection (the server has this identifier in cache).
  - `Cipher suites` - a list of cryptographic options supported by the client, in decreasing order of preference. Each item normally has a key exchange algorithm, an authentication algorithm, a bulk encryption algorithm (with key length and mode of operation), and a hash function. An example that has all mentioned algorithms is `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`. Other items may not have the authentication algorithm, such as `TLS_RSA_WITH_AES_128_CBC_SHA`.
  - `Compression methods` - a list of compression methods supported by the client, in decreasing order of preference. In practice, this methods are not used.
  - `Extensions` - used to inform the server of additional functionality supported by the client. Commonly used examples are the SNI extension, additional cryptographic parameters (supported groups, formats and algorithms) or the heartbeat extension.
- **Server Hello -** sent from the server to the client in response to the `Client Hello` message.
  - `Version` - specify the highest supported version by the client that is also supported by the server.

  - `Random structure` – current time followed by a random number (same as the client).
  - `Session ID` – if the session identifier of the `Client Hello` message is recognized in server's memory, then this value has the same value as the client's session identifier. In this case, the server also sends the `ChangeCipherSpec` message to resume the recognized session (skipping the full handshake procedure). In any other case, this value is empty and it means that a new connection needs to be established (full handshake).
  - `Cipher suites` – the selected cipher suite from the list sent by the client. The server will choose the highest cipher suite from the client's list that it also supports.
  - `Compression method` – the selected compression method from the list sent by the client. The server will choose the highest method from the client's list that it also supports.
  - `Extensions` – added in response to the extensions sent by the client. The server must only use the extensions that the client sent to him. If this behaviour is not met, the client must terminate the connection.

- **Server Certificate -** this message follows the `Server Hello` message. It conveys the server's certificate chain to the client, and is the only field of this message type. Note that this message is optional. In fact, it is possible to establish a connection between two anonymous parties, although is not common.
- **Server Key Exchange -** used to inform the client of additional cryptographic information, that was not on the Certificate, to agree on a *premaster key*. This message is optional, because not all algorithms need those additional parameters.
- **Server Hello Done -** means that the server is done sending messages to support the key exchange. This message does not contain any fields.
- **Client Key Exchange -** with this message, the premaster key is set. The content can be either the *RivestShamirAdleman (RSA)* premaster key encrypted with the server's certificate public key, or the client's *Diffie-Hellman (DH)* public key.
- **ChangeCipherSpec -** this message is not a handshake message, but one of the four protocols. This message contains a simple byte with value `1`, encrypted and compressed with the *current* state. Its unique purpose is to indicate the receiving side that the *current* state is now - at the time right after sending this message - overwritten with the *pending* state.
- **Finished -** this message is always sent after a `ChangeCipherSpec` message to verify that the key exchange and authentication processes were successful.
  - `Verification Data` – this field contains the hash of three different values: *master secret*, *finished label* and the *hash of previous messages*. The master secret is the final key used between the parties to perform encryption and decryption operations. Finished label is a simple string with the value *client_finished* or *server_finished*, depending on who has sent the message. The last value is the hash of all previous handshake messages, excluding `Hello Request` messages. Also, note that `ChangeCipherSpec` messages, `Alert` messages and `Record Protocol` messages are not handshake message types, so these are not included.

`Hello Request`, `Certificate Request` and `Certificate Verify` message types were not covered in the discussion above because they are both optional and uncommon. The first is sent by the server, at any time, to renegotiate security parameters. By others words, to begin the handshake anew (it must not be sent in the middle of a handshake). The purpose of the last two messages is to request the client for his certificate and to provide the correspondent verification with digital signatures, respectively.

### *ChangeCipherSpec Protocol*

As Figure 2.11 shows, this protocol message format consists of one single byte that has the value 1. Sending this message indicates the receiving side that the prior has changed the *current* state with the *pending* state.

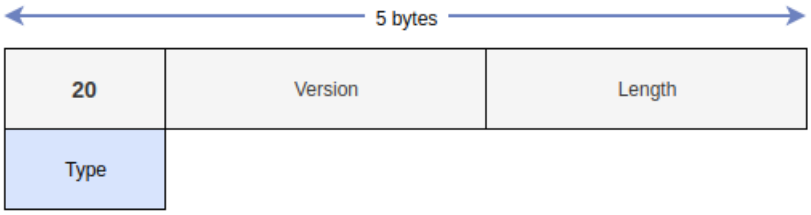



Figure 2.11: TLS ChangeCipherSpec Protocol message format [11]

### *Alert Protocol*

This protocol is intended to report warnings or fatal errors to the other entity. `Level` is the first byte of the message and indicates the level of severity (*warning* or *fatal error*). The second byte (`Description`) depends on the previous byte, and is the description of the alert (Figure 2.12). Note that it is not a description text (string), but an integer used to map its value to predefined descriptions.

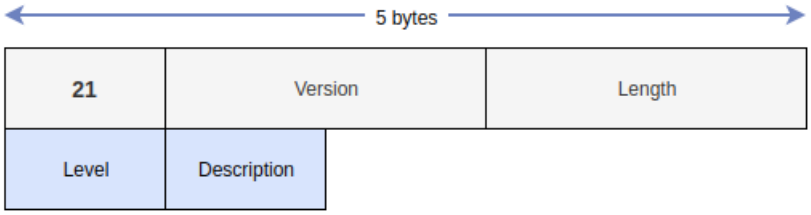



Figure 2.12: TLS Alert Protocol message format [11]

### 2.3.4 *Application Protocol*

Finally, the last protocol presented here is the one used to transfer application data. Data is treated transparently by the Record Protocol, which simply performs all previous described operations (encryption, compression, etc). Figure 2.13 shows that this protocol does not need any specific format, it just sends the data down to the Record Protocol.

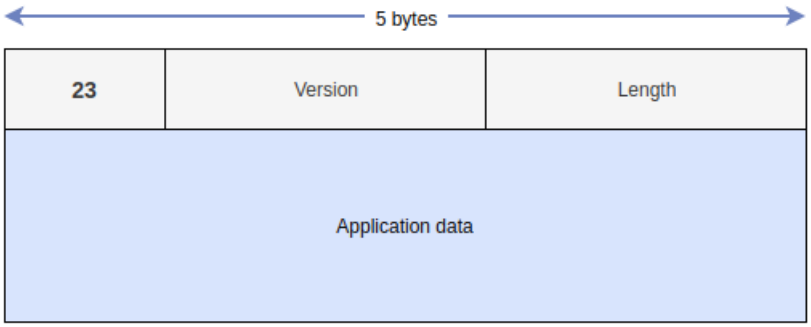



Figure 2.13: TLS Application Protocol message format [11]

## 2.4 TOR

Tor (or Tor Project), is a continuously growing open-source project that provides anonymity at network/transport layers in the TCP/IPv4 model. Its first release was in 2002 with the name *The Onion Router* (hence its actual abbreviated name). More specifically, Tor is a circuit-based low-latency anonymous communication service that provides perfect forward secrecy, congestion control, directory authorities, integrity checking, configurable exit policies, and a practical design for location-hidden services via rendezvous points.

Before explaining all those terms, some literature issues are discussed next. Back in 2004, a very complete academic document was published describing all Tor characteristics [12]. Since then, 13 design changes were made, described in three separated pages on the official website. Currently, there is no official paper gathering all information about how Tor works, the more recent being a draft from 2014 [13]. Because of that, many works are outdated. Nevertheless, this work takes into account all of the latest and up-to-date Tor specifications, as described in the *Design Documents* section in the official website [3].

Returning to the definition of Tor, it was said that the service is *circuit-based* because before sending any data, it creates a route of nodes through which the data will pass. Depending on the position in the circuit, each node can be called entry/guard, middle or exit. Further, instead of nodes, they can be called onion routers or relays. It is a *low-latency* service because it was deployed to be used in interactive or non-linear environments, such as web browsing. *Perfect forward secrecy* means that once the session keys are deleted, subsequently compromised nodes cannot decrypt old traffic (unique session keys are created for different sessions). Tor uses an incremental path-building design, where the initiator negotiates session keys with each successive hop in the circuit.

3 https://2019.www.torproject.org/docs/documentation.html.en#DesignDoc

*Congestion control* is also implemented to prevent onion routers from getting congested. *Directory authorities* are trustworthy servers responsible for retrieving control information to clients (e.g. list of onion routers that will make up the circuit). *Integrity checking* is performed in two stages. First, onion routers use the TLS protocol to communicate with each other. Second, specific Tor messages called *relay cells* contain an end-to-end checksum for integrity checking. *Configurable exit policies*, as the name suggests, is a feature that enables users to circumvent problems with service usage (e.g. choose the country of the last onion router in the circuit). *Location-hidden services* provide anonymous services to other users (users remain anonymous too), like e-commerce, news or illegal activities [12].

### 2.4.1 *Overview*

A typical architecture of Tor is depicted in Figure 2.14. This architecture is simplistic and hides much of the details about Tor's operation. At its core, Tor is simply a tool that can build paths given a set of routers [9].

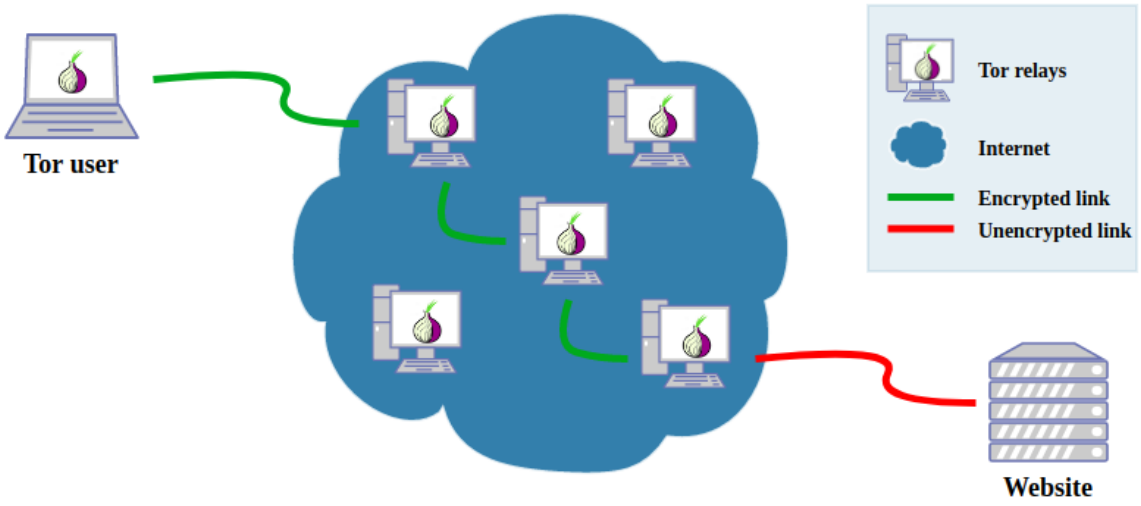



Figure 2.14: Tor architecture

The part corresponding to the unencrypted link (red line) is becoming more unlikely to occur. In fact, nowadays it is rare that the link is unencrypted because almost every website supports *Hypertext Transfer Protocol Secure (HTTPS)*. That is included in the diagram because it shows what fraction of the network is not controlled by Tor. The Figure also highlights that the website is the only element that can understand the messages sent by the user. For these reasons, Tor's users should not send personal information to the website. Therefore, most people use pseudonyms when using Internet forums or message boards from Tor.

Figure 2.15 depicts some more details. One relay is chosen more frequently, the guard node, which is highlighted in the Figure. In the one hand, this relay is a special trustworthy one that always acts as the entry relay, as long as it is not compromised or (periodically) rotated. On the

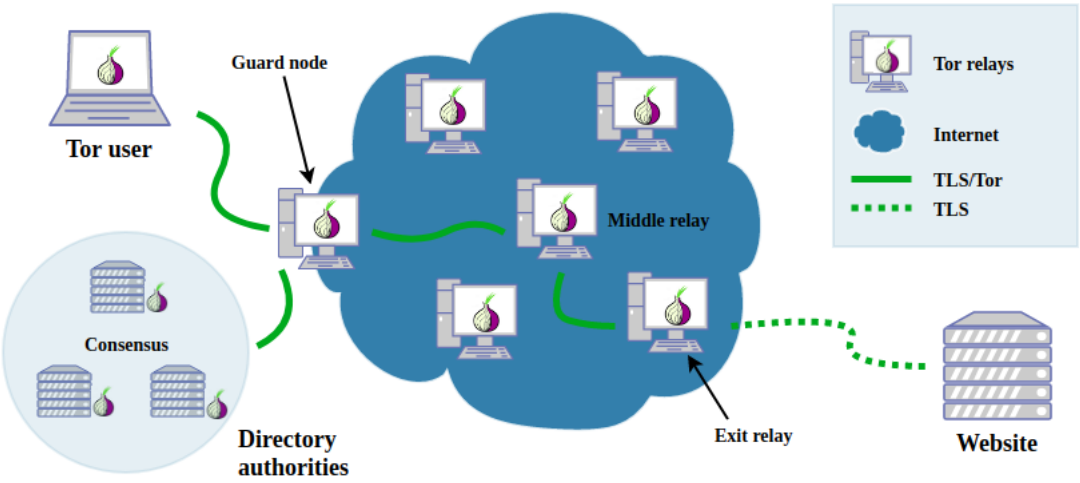



Figure 2.15: Tor architecture

other hand, *middle relays* and *exit relays* are chosen proportionally to their available bandwidth[4]. There are fewer exit relays than middle relays, because Tor allows volunteers to run either as middle or exit relays. The reason is that exit relays do not know what websites are being accessed by users, and so they are more vulnerable to attacks if the websites are (intentionally) compromised [13].

Another detail is the introduction of directory authorities, which are responsible for distributing control information across the (overlay) network. Basically, they agree on a consensus, a compressed document, so that each relay can check its validity. The user communicates with one of them through the guard node. Note that the previously red link's color was changed to reflect the fact that the link is usually TLS encrypted. Communications within Tor use their own defined encryption techniques (after the TLS), also depicted in the Figure.

### 2.4.2 *Tor Cells*

Before deepen in how Tor works, it is important to understand the information format exchanged between Tor nodes. This specific format has been given the name *Cell*. Each cell has a fixed-size of 512 bytes[5], a header and a payload. The header consists of a circuit identifier `circID` and a command `CMD`. The command defines what must be done with the payload. It can be one of three types: `control`, `relay` or `relay_early`. If an onion router receives a control cell, it must interpret the correspondent payload. Otherwise, the cell must be relayed to the next onion router. Figure 2.16 shows the overall structure of a cell. An important aspect is that circuit identifiers are connection-specific. This means that in the same built circuit, different end-to-end connections have different circuit identifiers between each other.

4 The bandwidth is constantly tested by directory authorities, to prevent attacks were relays claim to have more bandwidth than they really have, and are chosen more frequently [14][13]
5 Some cell's types, used for connection establishment, have a variable length.

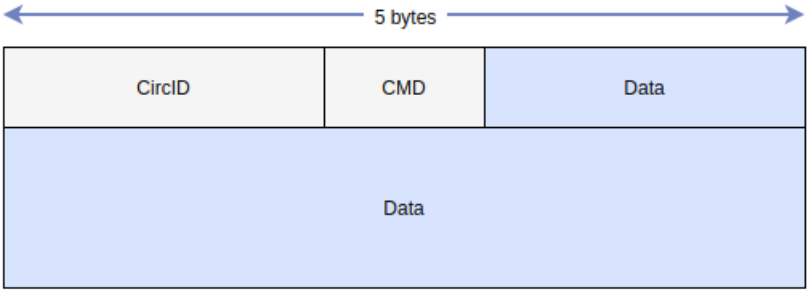



Figure 2.16: Overall cell structure [13]

*Control Cells*

The structure of this type of cell is the same as in Figure 2.16. In this context, the possible commands of *fixed-size* control cells are:

- `padding` – for keep-alive or link padding;
- `create` or `created` – to set up a new circuit;
- `create_fast` or `created_fast` – used to set up a new circuit to the first hop, without public key computation;
- `netinfo` – used to help nodes discover the current time and their own address;
- `destroy` – to tear down a circuit.

Possible commands for *variable-length* control cells are:

- `versions` – used for link-protocol negotiation;
- `vpadding` – variable length padding;
- `certs`, `auth challenge`, `authenticate`, and `authorize` – used for onion router/onion router and onion proxy/onion router authentication;

The data field depends on the command itself. For example, if the cell's command is `create`, then the receiving onion router knows that the data field carries the first half of the DH handshake protocol, encrypted with his onion router public short-term onion key. Thus, it can decrypt with his correspondent private short-term onion key.

*Relay Cells*

Relay cells need an additional header, as Figure 2.17 shows. This header contains a stream identifier `StreamID` to allow many streams to be multiplexed over a circuit, an end-to-end checksum for integrity checking `Digest`, the length of the relay payload `Len` and a relay command `CMD`. This last command should not be confused with the command that decides between a control or a relay cell.

The relay command can be one of the following:

- `data` – for data flowing down the stream;
- `begin` – to open a stream;

- `begin dir` – to open a local stream for directory information;
- `end` – to close a stream cleanly;
- `teardown` – to close a broken stream;
- `connected` – to notify the onion proxy that a relay begin has succeeded;
- `extend` and `extended` – to extend the circuit by a hop, and to acknowledge, respectively;
- `truncate` and `truncated` – to tear down only part of the circuit, and to acknowledge, respectively;
- `sendme` – used for congestion control;
- `resolve` and `resolved` – used for anonymous *Domain Name System (DNS)*;
- `drop` – used to implement long-range dummies.

The entire contents of a relay cell - except for the `CircID` - are encrypted or decrypted together as the cell moves along the circuit, using the 128-bit *Advanced Encryption Standard (AES)* cipher in counter mode to generate a cipher stream.

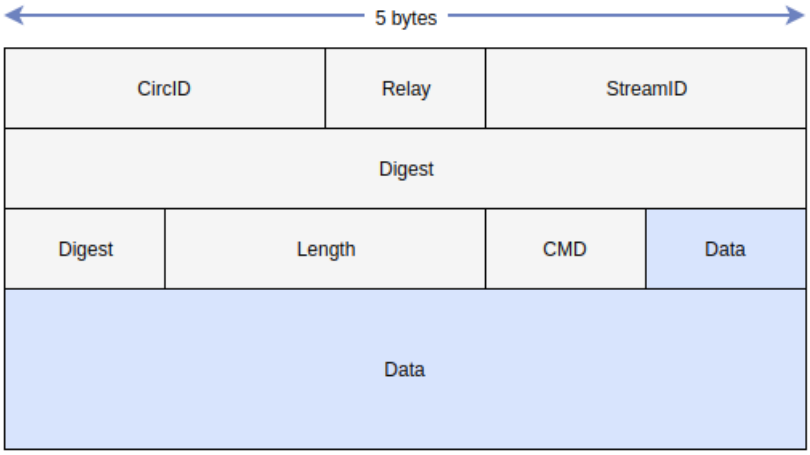



Figure 2.17: Relay cell structure [13]

*Relay_early Cells*

These cells work similarly to relay cells but are distinguished to enforce the maximum path length.

### 2.4.3 *Tor Operation*

In order to describe Tor's operation in more detail, three different logic steps are explained:

- Fetching information from directory authorities
- Building a circuit
- Relaying traffic through the established circuit

### *Fetching information*

The idea behind directory authorities is very simple. Basically, each one creates an abbreviated version of each descriptor they recommend. The descriptors are a summary of the router's exit policy, and the router's current public onion key. Of course, verification is also done. Servers exchange each one's information periodically to agree on a consensus and sign the document at the end to prevent the case where a single compromised directory authority can advertise false descriptors. This solution, retrieving control information from static IPv4 addresses, makes the blocking of Tor traffic very easy for service providers or censors [9]. To prevent this, the Tor team emerge with the idea of *Tor Bridges*, special Tor nodes which are not published in the directory authorities, and can be used as entry points to the network (both for downloading the authority information and also for building circuits). This hardens network blocking but is not a perfect solution either. Although it makes blocking more difficult, it is a matter of time until censors can detect all of these IPv4 addresses [9]. However, how do clients fetch these special nodes, if they cannot use any directory authority? From the official documentation, it is suggested to use e-mail exchange, social networks interaction, or even to speak in person to someone that knows about some Tor bridge. After getting one bridge that can reach the overlay network, it is possible to get a few more (some can be blocked at any time, and this approach prevents the client to be fully disconnected from the network again).

### *Building a circuit*

Assuming that routers' information has been fetched, an onion proxy needs to choose his path. Remember that directory authorities deliver not only three routers but a set of recommended routers, so the client can pick only a subset of them (normally three). To this end, as stated in the official Tor website, nodes are chosen proportionally to their bandwidth, as weighted by an algorithm to optimize load-balancing between nodes of different capabilities [6]. Now, suppose that the client chooses three onion routers with the previous algorithm. By incrementally negotiating session keys with each onion router (e.g. key1, key2 and key3) the client successively encrypts the message. For example, a message `m` is encrypted as follows, `E( E( E(m, key3), key2), key1)`, where `E(message, key)` is the encryption algorithm. Each onion router in the path then strips off a layer of encryption, until the exit node decrypts the last layer and becomes able to redirect the original message's request. Figure 2.18 illustrates this process with only two onion routers (in real operation there are at least three).

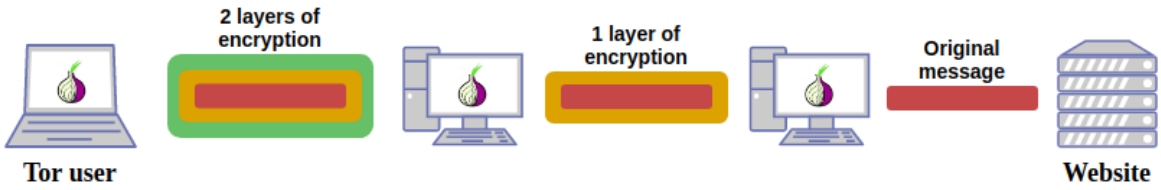



Figure 2.18: Tor Encryption Layers [13]

6 https://blog.torproject.org/top-changes-tor-2004-design-paper-part-2

*Relaying traffic*

Finally, traffic can be relayed from the onion proxy to the final destination, through the circuit just built. When the first node (guard node) receives a message from the client, it checks the corresponding circuit this message belongs to, and, because it knows the next hop associated with this circuit, it simply sends this message to the IPv4 address associated with. Remember that each host has a different circuit identifier, although all belong to the same circuit. This process continues until the message reaches the final onion router, the only one that can decrypt the traffic, that checks the payload, and makes all necessary requests. Thus, the resource that responds to these requests (server) only knows about this last onion router IPv4 address.

### 2.4.4 *Tor example*

In this section, a step by step of Tor operation is explained. This will help to understand all the details about datagrams, segments, and messages contents. For this, it is assumed that the client has just downloaded an official version of the Tor Browser and, because of that, has not fetched any information from the directory authorities.

In the first step, the client will choose a directory authority that comes hardcoded in the Tor source code [9]. However, it will not communicate with that server directly. Before that, a guard node is used, an onion router that the client chooses as the first hop for all circuits (which comes hardcoded as well). This guard node is periodically rotated (4-8 weeks)[13]. The client will establish a TLS connection with that node.

After the TLS handshake, both parties will start exchanging Tor cells through this connection. Note that they will not start sending application data just yet. Some security parameters need to be exchanged, specifically the two secrets between them (one secret for each direction). To achieve this, the client creates a control cell. The payload of this cell contains the first half of the DH handshake, encrypted with the guard node public RSA key. The guard node will respond with a created cell containing the other half of DH handshake, along with a hash of the negotiated session key (this is not the final key, but instead is used to derive four other keys: one for each direction for AES, and one in each direction for integrity) [13].

After receiving the created cell, the client sends a relay cell with the *Hypertext Transfer Protocol (HTTP)* request to the directory authority chosen. Upon receiving this cell, the guard node looks up the corresponding circuit and decrypts the relay cell with one of the two session keys. Then, a plaintext HTTP request is made to the directory authority to fetch the current state of the (overlay) network. After receiving the response from the directory authority, the guard node wraps the response in a relay cell, encrypts it with the other direction session key, and sends the cell back to the client. Finally, the client decrypts the cell and obtains a list of available onion routers.

To this point, the client has a guard node with a Tor connection already established, and a list of some onion routers. Now, the client executes an algorithm[7] that will return a path of at least three of these routers.

To extend the circuit by one more hop, the client creates a relay extend cell, specifying the address of the next chosen onion router and, as with the first hop, the first half of the DH handshake

7 https://blog.torproject.org/top-changes-tor-2004-design-paper-part-2

encrypted with the chosen onion router public RSA key. The onion proxy encrypts the cell using the negotiated AES key between himself and the guard node. The client sends the relay extend cell to the guard node. After receiving the cell, the guard node decrypts it with the AES symmetric key, creates a control cell containing the received encrypted DH data, and sends to the received address (the next onion router). After receiving that cell, the onion router will respond with the other half of the DH in a created cell (basically the same procedure between the client and the guard node). At some point, the guard node receives that cell, wraps it in a relay extended cell, and sends it back to the client. Receiving that type of message, the client knows that the circuit was successfully extended by one more hop. The client has negotiated two symmetric keys with the two first hops in the circuit. The operation to extend the circuit through more hops is the same as described above.

After completing the circuit, the onion proxy generates its messages as normal, typically HTTPS, and encrypts it with each negotiated key. When the message arrives at the last node, it will perform the last decryption, understanding the message and making the necessary requests. This process is transparent to the user, and the last node cannot understand the contents of the HTTPS data. If HTTP is used instead of HTTPS, this last node can understand every message sent by the user.

## 2.5 Intrusion Detection Systems

An IDS is a security software program that monitors network packets or local files for malicious activities, policy violations or unsuspected behavior. It can be a dedicated device because specific hardware increases processing speed.

These systems are commonly used by *Internet Service Providers (ISP)*, organizations or institutions in conjunction with a firewall. While the firewall prevents undesired traffic from entering the network, the IDS complements this task by monitoring traffic that may bypass firewall's rules (for example, if there is no rule for a specific malformed packet).

The literature defines different categories of IDS depending on specific attributes. Depending on the system's role, it can be a *Host-based Intrusion Detection System (HIDS)*, when it monitors files in the local host operating system, or a *Network-based Intrusion Detection System (NIDS)*, when it monitors network traffic.

Depending on the system's reaction, an IDS can be passive or reactive. Passive systems only report information after a security breach. The latter, also known as *Intrusion Prevention System (IPS)*, tries to adjust local configurations in real-time after a rule violation.

The last category depends on the detection approach. The two common approaches are signature-based and anomaly-based. Signature-based systems have a predefined set of rules (called signatures) that are compared with the information being monitored. If a match is found, the system generates an alert. Note that it requires previous knowledge of the possible attacks to generate accurate signatures. Even if a signature is matched, it may not be the result of an attack, so a false alarm is generated. Also, note that every packet must be compared with an extensive collection of signatures, and the system can easily run out of resources [3][5].

The other approach, anomaly-based, is often associated with machine learning. The idea is to train the system with the expected normal behavior, creating a profile. Then, when in operation, any unusual behavior that exceeds some threshold deviation will trigger an alarm. Thus, as opposite to signature-based systems, attacks not yet known/documented can be detected. Despite the

mentioned drawbacks, most IDS deployments are primarily signature-based, although some include anomaly-based features [5].

Some newly different categories have been suggested in the literature, as well as different combinations among them. For example, some hybrid approaches join host-based and network-based in a single IDS. Other systems are simultaneously passive and reactive, known as IDPS (Intrusion Detection and Prevention System) [15].

### 2.5.1 *Open-source IDS*

To be able to detect the Tor traffic, the choice fell on a signature-based system. As will become more clear in the next chapter, this choice relies on the knowledge in advance about the traffic characteristics to detect. There are many open-source IDS to choose from, such as *Snort*, *Bro* or *Suricata*.

Here, the IDS to choose from is not a major concern, as long as it allows one to test the signatures desired. To this end, it was decided to choose Snort. Along the fact that it is the de-facto open-source IDS, it was developed in an open-source operating system. These facts enhance its diffusion and exchange of knowledge among all interested.

In all Snort's features, the more significant to this work are the support of stateful inspection, the capability of acquiring the traffic (to inspect) in different modes, and the flexibility in its configuration and logging systems.

### 2.5.2 *Snort*

Snort[8] is a free open-source network-based IDS and IPS, created in 1998 by Martin Roesch. He latter founded Sourcefire, the company behind Snort development, which was acquired by Cisco in 2013.

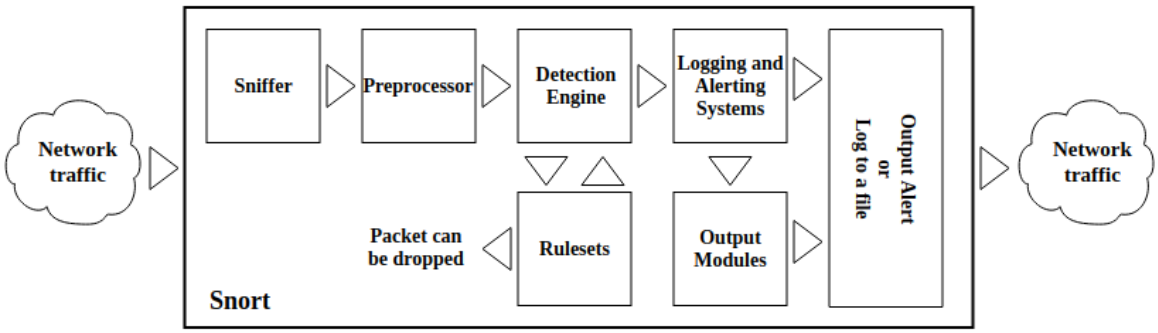



Figure 2.19: Snort architecture [16]

Snort can be executed in three different modes:

- `Sniffer mode` - reads network packets and displays them on the screen;
- `Packet logger mode` - logs the packets to disk;

8 https://www.snort.org/

- `Network Intrusion Detection System mode` - performs detection and analysis of network traffic.

In the present context, the interest is in the last mode, because it allows not only monitoring network packets but also comparing them with a predefined set of rules[9]. Snort comes with a lot of different configuration and filtering options, supports different packet acquisition libraries, provides rules already tested (called community rules), among many other features [16].

Attending to the focus of the work, two more topics need to be discussed, namely understanding how to create rules in Snort and understanding the format of logs to interpret the generated alerts.

### *Rules Format*

The syntax of Snort rules is very simple. Listing 2.1 represents the syntax that the rules must comply with, while Listing 2.2 materializes that syntax and makes it easier to understand.

```
action protocol ip_address port direction ip_address port \
    (option_keyword_1:value; option_keyword_2:value; \
        option_keyword_3:value ...)
```

Listing 2.1: Snort rules syntax

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rules action (alert, drop, etc), protocol, source and destination IPv4 addresses and netmasks, and the source and destination ports information. The text up to the first parenthesis is the rule header (the \ character is used to break lines). The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken (section enclosed in parenthesis). The words before the colons in the rule options section are called option keywords.

```
alert tcp 192.168.1.0/24 any -> 183.21.34.234 80 \
    (flags:S; msg:"SYN packet";)
```

Listing 2.2: Snort rule example

The example shows a rule that triggers for each attempt of initiating a TCP connection (SYN flag) to the IPv4 address 183.21.34.234, from the subnet 192.168.1.0/24. When triggered, the alert shows the message `SYN packet`.

### *Alerts and logs*

When a packet reaches the detection engine, it will be compared with rules defined in several files (`.rules`). In the configuration file, one specifies which ones to include in a specific execution. The number of comparisons performed depends on whether the packet will match a rule or not. If there is no rule matching the packet, Snort will iterate through all of them, and will not trigger any action. Otherwise, after a match, Snort will trigger the action (`drop`, `alert`, etc) specified in that

9 In Snort's jargon, rules are the same as signatures.

rule. The detection engine will match only once, meaning that it will stop comparing a packet to the remaining rules after a first match. When a packet matches a rule, it is said that a *violation* has occurred.

Listing 2.3 shows an alert log generated after a violation. By default, Snort logs do not have all this information. Logs like that in the Listing are generated after adding the line `output alert_full: alert.full` to the configuration file. It simply means to log (output) the alerts in the alert_full format, to a file called alert.full. It is possible to check that the rule's generator (`GID`) has the value 402 and the rule's code inside that group (`SID`) has the value 7. The `GID` is used to categorize different kinds of rules. For example, backdoor rules have a different `GID` than SNMP rules. The `SID` is then used inside each group to give each rule an identifier. It is also possible to inspect the alert's description, followed by its classification and priority (according to whom created the rule). Finally, a dump of the packet that generated the alert, which includes a timestamp and network/transport layer information.

```
[**] [1:402:7] ICMP Destination Unreachable Port Unreachable [**]
[Classification: Misc activity] [Priority: 3]
10/22-11:15:04.559114 172.16.16.24 -> 172.16.16.16
ICMP TTL:128 TOS:0x0 ID:41606 IpLen:20 DgmLen:216
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
172.16.16.16:53 -> 172.16.16.24:56454
UDP TTL:64 TOS:0x0 ID:8716 IpLen:20 DgmLen:188 DF
Len: 160  Csum: 25868
(160 more bytes of original packet)
** END OF DUMP
```

Listing 2.3: Snort alert example

## 2.6 RELATED WORK

This section presents the literature review. Only Tor's related works are included, but some of them may deviate from the aim of this work. Nevertheless, all of them are recent and can be used to perceive the different areas of research about Tor.

In [17], machine learning techniques are used for traffic classification, in three different anonymous tools: Tor, JonDonym and I2P. They measure three different levels of granularity to distinguish between anonymous tool, traffic type and application. The work was based in the *Anon17* dataset[10]. The results showed nearly 100% accuracy in the first granularity level (anonymous tool), around 85% in the second level, and 67% in the last (application). The percentage values presented here coincide at the moment after receiving the flows' eighth packet. This choice (eighth packet) is comprehensible because the beginning of a flow contains the negotiation of parameters, and requests between client and server or between peers, with protocol-specific headers and message sequences [3]. Despite this, the authors also show the maximum achieved accuracy and F-measure, along with the number of packets needed to achieve those results. Additionally, the effects of feature importance

10 https://web.cs.dal.ca/~shahbar/data.html

and temporal-related features to the network are investigated [11]. This work has the disadvantage of a closed world assumption.

Another work [18] demonstrates the presence of *hot exit points* in Tor. These are exit-nodes agglomerations, controlled by some ISP, that are almost always used, despite the existence of other exit-nodes options. The results are based on 1.5 years of recorded data, and, by our previous discussion about Tor, is a threat to anonymity (ISP can correlate traffic more easily).

A different Tor design is proposed in [19]. By introducing group signatures, a cryptographic technique, it is possible to distinguish between legitimate and illegitimate users. When some malicious action is performed by a user, it is possible to block or denounce him. This approach is not perfect. Although relay volunteers and ISP gain trust in the overlay network, benign users have to trust the entity in charge of blocking or denouncing them (basically the same problem of mixing proxies).

One more contribution comes from *Identifying TLS abnormalities in Tor* [11]. The work begins by identifying TLS characteristics in Tor (e.g. certificate extensions). Then, Snort rules are created according to the characteristics mentioned above, which could effectively identify the presence of Tor traffic. However, in the time of that writing, Tor supported TLS 1.0. Further, with the introduction of TPT, TLS characteristics are obfuscated. For example, previous versions of Tor used to only support the strongest cipher suites in the TLS Handshake. Because famous websites supported a bigger collection of cipher suites, checking the supported number of cipher suites could suggest the presence of Tor.

The work in [20] begins by identifying the most known attacks that could be used to de-anonymize Tor circuits. The two outlined categories of attacks are traffic correlation and webpage fingerprinting. Further, the vulnerable identified areas are the guard-node selection & rotation algorithm, the inter-cell transmission timings and other traffic metrics (cell order, amount, interval, size and direction). Then, the authors research these areas to collect the proposed and already-implemented countermeasures that could enhance Tor resistance in the three mentioned vulnerabilities. Finally, the process of evaluation is accomplished by comparing each countermeasure with a set of (security) requirements previously defined (following the *MoSCoW* method).

A longitudinal study about the Tor network is presented in [21]. The work is based on a passive analysis of TLS traffic over more than three years in four large universities. The results show that it is possible to identify Tor, specifically through some information present in X.509 certificates and other exchanged parameters within the TLS handshake. Despite this, it is assumed that Tor's detection will remain an arms-race.

## 2.7 SUMMARY

This chapter has reviewed concepts needed to follow the rest of the work. It started by examining the discipline of traffic classification, its evolution and the state-of-the-art. Anonymity systems were also introduced, as the architectural ways employed to provide anonymity to users.

TLS and Tor were described in the following two sections. In the one hand, TLS is the protocol that Tor relies on to provide most of its security requirements. On the other hand, Tor was the chosen case study of an anonymity system. Given their importance in this context, each one was described with some detail.

11 The number of features goes to a maximum of 74, which are then correlated.

The chapter ended with a discussion about IDS and the related work about Tor. After a brief description about IDS systems, a specific IDS, Snort, was chosen and explained to allow one to materialize the detection of Tor traffic. The related work summed up the most recent contributions and studies about Tor.

# 3

# METHODOLOGY

This chapter introduces some more concepts discussed in the previous chapter, but this time already applied to Tor. First, the considered assumptions and a brief discussion about the different contexts of monitoring observers. Second, a study of Tor traffic characteristics, through an individual analysis of each layer in the TCP/IPv4 model. Finally, the Snort rules created according to the traffic characteristics studied.

## 3.1 EXTERNAL OBSERVERS

### 3.1.1 *Observer positioning*

This section is dedicated to discuss the observer's position or which portion of traffic it can monitor. In terms of positioning, note that it can be:

- between the onion proxy and the guard node;
- between two onion routers (or a guard node and an onion router);
- between the exit node and the final destination.

Figure 3.1 illustrates these three different cases, that are discussed next.

#### *Onion proxy and guard node*

An external observer eavesdropping any link between the onion proxy and the guard node can infer some information. First, the (partial) location can be exposed through the inspection of the IPv4 addresses involved in the connection. Second, it is possible to conclude if the client is using Tor. Either inspecting the IPv4 address and comparing it to known guard nodes[1], or inspecting the TLS handshake (assuming that the connection was not already established).

Note that by accessing different websites, the onion proxy will remain connected to the guard node, and an observer, in this case, can only monitor network metrics.

1 Figure 3.1 does not illustrate the presence of Tor Bridges. In that case, the onion proxy connects to a bridge before the guard node and can obfuscate itself from his regional ISP. That is, the regional provider could only conclude that the onion proxy was communicating with another peer.

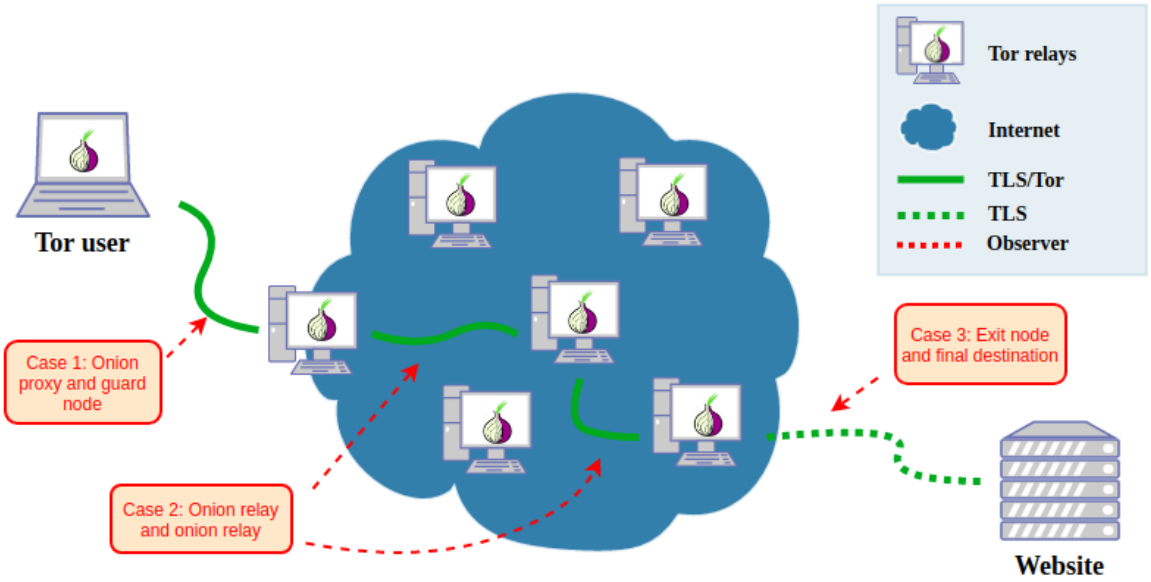



Figure 3.1: Observer different positions in the Tor network

### *Two onion relays*

This case and the first have different traffic patterns. The TLS handshakes are slightly different, but an external observer cannot distinguish them. One might think that it is easy to distinguish the two different handshakes, because in the first case only one party authenticate himself (the guard node), and in this case both need to authenticate (prevent impersonation attacks). Tor handles this by performing a TLS handshake first, always as a server-only authentication handshake, and then an inner handshake to complete the authentication. To better illustrate this, imagine that two onion relays need to communicate. They will start by performing a normal client/server TLS handshake, and only one of them (the next in the circuit) will authenticate himself with his certificate. Only after that, inside the TLS session just created, will they complete the authentication in an inner handshake, specified by Tor (not a TLS handshake). In this inner handshake, they will agree in a common version, exchange certificates, and other parameters. To an external observer, the outer handshake is the same for those two cases.

After the handshake phase, traffic patterns can be slightly different. Only the first case (onion proxy and guard node) uses connection-level padding. Any other connection does not use it (a connection to a bridge is treated as the first case[2]). There are change proposals about circuit-level padding between onion relays but are not yet implemented. In connection-level padding, cells carry the same value as always, but new dummy cells are introduced to obfuscate traffic from external observers. So, in theory, this and the last case have different traffic patterns from the first.

### *Exit node and destination*

This last case, from the exit node to the final destination, despite being a traditional HTTP/ HTTPS connection, can be measured to possibly detect the Tor presence. One way is to inspect the IPv4

2 https://gitweb.torproject.org/torspec.git/tree/padding-spec.txt

addresses involved in the connection and compare them to a list of known Tor (exit) nodes. The other possible way is to measure network metrics and infer that it is a Tor exit node because the delay of packets is higher than the normal. In this case, the packets' length is not fixed. The size of messages follows the normal operation of the HTTP protocol, but the exit node readjusts them to fit the fixed demanded size (512 bytes) in the Tor network.

This might be helpful, for example, if a website administrator does not want Tor users to access his website. One possible reason for an administrator to want to block Tor under these conditions is that the information collected (IPv4, interactions, etc) is no longer related to the original client's IPv4, and is now related to the exit node. Worse than that, a different client may use the same exit node, but they look the same to the website because both requests came from the same address.

### 3.1.2 *Observer portion*

In normal conditions, a passive observer can only monitor one link. However, monitoring more than one (overlay) link in Tor can be only achieved if that observer spans different countries. In fact, Tor assumes that does not have a defense against a global adversary [13]. By global adversary is meant a passive observer that can monitor more than one link, thus correlating flows and inferring the circuit that an onion proxy is using.

Of course, one global adversary can be made of smaller adversaries working together. Being it in real-time or in recorded logs, it is possible to aggregate that information and correlate it to know what website a client was accessing through Tor.

### 3.1.3 *Assumptions*

To test the detection of Tor, some facts are assumed to simplify the evaluation process. First, Tor is tested only in the first observer's position case (between the onion proxy and the guard node). The traffic sniffed in the computer executing the onion proxy is the same as the traffic that an eavesdropper can sniff between the two points. Second, the detection approach was developed taking in mind that only a single point in the architecture could sniff traffic (small adversary between the onion proxy and the guard node). Third, any change that could change Tor normal operation was not considered. This includes the usage of bridges, TPT, or any software's change.

## 3.2 TRAFFIC CHARACTERISTICS

The presence of Tor traffic can be detected by examining its characteristics in detail and create rules that match all of them. The main advantage of this approach has to do with its higher accuracy. Nonetheless, an increasing number of rules, or a more extensive single rule, will also increase the system's workload. In an overloaded IDS, that already needs to compare thousands of signatures, looking for hundreds of different protocols, the worthiness of introducing more rules about Tor can be questionable. Even if no Tor traffic is being generated, for each packet Snort will search all the rules anyway, looking for a match (this was explained in subsection 2.5.2).

Another solution is to choose as few characteristics as possible which can correctly classify Tor. Because browsers and many websites have common traffic characteristics, Tor can be distinguished among them. This leads to a trade-off between rules completeness (accuracy) and workload. Generic rules are computationally less expensive and will detect Tor, but possibly generating false positives.

An external observer can inspect packets' payloads, but those are encrypted and do not retrieve any useful information (at least individually). Because of that, one can only inspect lower levels, such as network, transport and session layers. Simultaneously, measuring some network metrics can help in detecting Tor traffic. In cases where TPT is being used, it is more complicated (by default it is not used). Perhaps the most important metrics in this context are the length of packets and the delay. The former is useful because Tor uses fixed-size messages with 512 bytes (connection establishment cells are variable-sized [3]). The latter (delay) can suggest the presence of Tor, because the values will typically be higher than the normal (packets have to traverse onion relays across different countries). It should be used in conjunction with other metric/rules for completeness (a network congestion could produce a false positive).

As a brief note, in subsection 2.3.1, TLS is said to be between transport and application layer, as stated in the official *Request For Comments (RFC)* 5246 [10]. However, in Snort, TLS is included in the application data (payload), and so the discussion about the payload being encrypted is not absolutely right in this case. Because of that, from now on, TLS handshake is considered to be application layer data.

Having discussed overall considerations about Tor traffic characteristics, it follows a discussion of how Tor treats each protocol individually.

### 3.2.1 *Network Layer*

At the network layer, the IPv4 is the first protocol worth to be inspected. It is known that some ISPs are already blocking static IPv4 addresses used by directory authorities, onion relays or guard nodes (it was this fact that led to the introduction of Tor bridges) [9].

There are lots of websites that show the current active onion relays, their associated IPv4 addresses, geographical location, and sometimes the volunteer's personal information (Twitter profile).[4] Further, even with the introduction of bridges, it is assumed that there is no magic bullet for their discovery. Although it is more difficult to enumerate all the bridges IPv4 addresses, it is a matter of time until censors enumerate them all [9].

### 3.2.2 *Transport Layer*

Tor Browser currently uses random ports to communicate with the guard node. Onion proxies use the port that guard nodes chose. The same applies to fetch network information through other relays (up to 3 connections). Each relay can optionally act as a directory authority to help in the distribution

3 In Wireshark, one can check this by examining the TLS Record Protocol Encrypted Application Data field. Successive packets have 538 bytes. The length is not exactly 512 because the connection recorded used *Galois/Counter Mode (GCM)* (*Authenticated Encryption with Associated Data (AEAD)* cipher), that will produce larger encrypted blocks than the received (un)compressed block [10].

4 `https://torstatus.blutmagie.de/index.php?SR=Bandwidth&SO=Desc`

of the consensus document, but the ports are randomly chosen as well. The randomness of ports at this layer makes it useless from a signature-based IDS point of view. However, an anomaly-based IDS could probably deduce some information if some ports were used more than others.

### 3.2.3 *Session Layer*

In the case of TLS, the followed approach is simple. The objective is trying to detect fields and parameters that are different from other TLS flows (web browsers). By comparing the Tor Browser with others, one can start ruling out other TLS sources until there is only the Tor Browser left.

According to [21], there are lots of fields in the TLS protocol that can be used to detect Tor. This work takes that list as a starting point to discuss some of the TLS fields that will be used to create the rules. To each field, there is a simple discussion that justifies the reason why each one is or is not used in the rules.

#### *Client Hello*

Starting with the `Client Hello` handshake message, the first useful parameter to inspect is the TLS version. Because current browsers (not all of them) already support TLS 1.3, every website connection within the browser will use an extensive list of TLS extensions in `Client Hello` messages. That is how backward compatibility is handled. A client that supports version 1.3 sends way more extensions than a client that only supports version 1.2. If the server also supports that version (1.3), it will reply to that specific extensions. If it does not, it will reply as it would for version 1.2 or below. Inspecting the TLS `version` field is useless, because both versions use the value `0x0303` for backward compatibility.

Because Tor nodes do not resume sessions, the `Session ID` field can be helpful. Services commonly used can have these values in cache. When both client and server have the same values, the full handshake process becomes a partial handshake. For example, when the network being monitored only uses known services that usually resumes old connections, this information can be profitable in ambiguous cases.

Regarding cipher suites information, each browser supports the list of cipher suites desired. A major exclusion parameter is the `ciphersuites length`. If this value is the same between Tor and any other browser, one can compare the actual `ciphersuites list` (the length can be the same, but the list can be different). Remember that each client sends the cipher suites list in decreasing order of preference.

The `compression methods` field is not commonly used. Neither the studied browsers nor the Tor support compression methods.

`Extensions` can vary among browsers. In the one hand, they can be different because some browsers do not yet support TLS 1.3, and some extensions were created specifically for that version. On the other hand, even if the versions match, each one can have different design choices. At the moment of this writing, IANA has 45 specified extensions. Additionally, it reserves a range of values for private use[5]. One downside of using the `extension length` field is that, unlike the `cipher suites length` field, the length differs for each extension. Each cipher suite is identified

5 https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.txt

by a two-byte value, which does not happen in the case of extensions. For example, the `hearbeat` extension has 1 byte, and the `signature algorithms` extension is a list containing two-byte values (a list of 15 algorithms gives 30 bytes).

Table 3.1 compares two of the discussed fields in the `Client Hello` message for Tor and the most used browsers according to *w3schools* [6].

| Browser | Ciphersuites | Extensions |
|---|---|---|
| Tor | 14 | 6 |
| Chrome | 17 | 17 |
| Safari | 23 | 8 |
| Edge | 19 | 10 |
| Firefox | 18 | 14 |
| Opera | 17 | 17 |

Table 3.1: Client-side TLS parameters

Opera and Firefox generate some TLS 1.3 traffic, because both use some Google services (such as Google Analytics).

### *Server Hello*

The `Server Hello` handshake message also carries useful information. However, the scope is bigger in this case. The discussion is not about the client-side (browsers), but the server-side. This message should only be inspected to complement the prior, because the number of different servers is very large, and each server has to follow the client behavior. Relying on this single message could give many false positives.

To accomplish what was just said, the previous message needs some sort of state. More advanced firewalls and IDS/IPS provide stateful control of traffic flows [2] (as is the case of Snort). In real operation, these messages can be separated by hundreds of milliseconds. In that interval, depending on the link's capacity and the current traffic, many more packets can arrive. A choice has to be made between maintaining flows' state and waiting for this second message, or use this message individually. Another option is not even to consider this message alone if it is shown that gives to many false positives. Again, it is a trade-off between accuracy and performance.

The discussion about the `version` field applies here the same way it applies to the `Client Hello`.

A typical server will try to choose the securest cipher suite supported that the client also supports. This remains true in Tor.

The `extensions` field, taking the previous discussion about `Client Hello` fields, can be a major field in the detection process. As it was said, the `version` field is the same for TLS 1.2 and 1.3. Servers know that a client is using TLS 1.3 because the client uses some specific extensions in the `Client Hello` message that are only used in that version. Because of this behavior, if servers want to communicate with the client via TLS 1.3, they will respond with an unusual extensions list (unusual

6 https://www.w3schools.com/browsers/default.asp

to a TLS 1.2 communication). Even if the communications uses TLS 1.2, Tor chooses to use an extension list that usually has a length of 13 or 18.

### *Certificate*

Another important message is the one that carries the `Certificate`. In previous versions, Tor's use of fixed two-certificate chains was a giveaway to anyone wanting to block it [13]. Now, Tor nodes send their own certificates, signed by other authority nodes. An external observer can inspect all the fields, but the `subject name` (whom the certificate belongs to), the `issuer name` (who signed) and the `validity time` are important fields to distinguish different websites.

The subject and issuer names are important because Tor does not use them as usual. Typically, a website uses the subject field in the certificate as a traditional *Uniform Resource Locator (URL)*, or at least in an understandable way. For example, just by opening Goggle Chrome (with a Google account signed in) a few connections are established. Each connection uses TLS 1.3 and has its own certificate. The certificates' subject names are all related to the Google name, and all of the certificates' issuer is `Global Trusted Sign` (the trusted entity that issued them).

In the case of Tor, two characteristics stand out. The first is that both name fields, subject and issuer, are random strings (e.g. `www.e635txx4ywriqq.net`). Second, the issuer's name is not from a known trusted entity (like Digicert). Instead, it is another random string, that is issued by Tor's directory authorities. So, Tor does the proper *Public Key Infrastructure (PKI)* validation with its own trusted *Certificate Authority (CA)* system. Nodes will automatically generate X.509 server certificates, which they periodically rotate. As it turns out, Tor's current certificate algorithm leaves them identifiable through pattern matching, enabling passive observers to distinguish Tor connections from other TLS connections. Tor nodes authenticate each other by validating the exchanged certificate's signature with the public keys from the directory authorities [21].

Another field is the `Validity Time`, that has two values, `Not Before` and `Not After`, indicating the period in which the certificate is valid. Here, the interest does not reside in the specific dates, which vary from server to server, but on how long the periods are. Examining some websites can give us insight about commonly used times, and help in distinguishing Tor.

Table 3.2 presents relevant fields and parameters of `Certificates` messages from the 10 most used websites. The values are from *Qualys SSL Labs* [7], a website that can test TLS parameters of clients and servers.

Note that the values from Table 3.2 can assume different values. A website can retrieve a slightly different certificate chain depending, for example, on the browser being used. As it is possible to check, Tor nodes only retrieve a single certificate in the chain, while other websites retrieve at least 2. Also, as previously stated, issuer names are from globally trusted companies, except for Tor. Only one `validity time` has the same time-space of Tor, which is the first certificate offered by Facebook. What should be retained is that by correlating more than one field it should not be difficult to detect a Tor's certificate (chain).

7 https://www.ssllabs.com/

| Servers | Certificate Validity time | Certificates | Issuer |
|---|---|---|---|
| Tor | 3 months | 1 | (Random) |
| Google | 1 year<br>10 years | 2 | Global Trusted Sign |
| Facebook | 3 months<br>10 years | 2 | DigiCert |
| Baidu | 1 year<br>10 years | 2 | GlobalSign |
| Wikipedia | 1 year<br>11 years | 2 | GlobalSign |
| Tencent QQ | 1 year<br>10 years | 2 | DigiCert |
| Taobao | 1 year<br>10 years | 2 | GlobalSign |
| Tmall | 1 year<br>10 years | 2 | GlobalSign |
| Yahoo! | 6 months<br>15 years | 2 | DigiCert |
| Twitter | 1 year<br>15 years | 2 | DigiCert |
| Amazon | 1 year<br>15 years<br>5 years | 3 | DigiCert<br>VerySign |

Table 3.2: Server's certificate parameters

## 3.3 SNORT RULES

Taking the previous details into account, the Snort rules were created according to the described characteristics. The first rule is listed in Listing 3.1. Expressed in words, this rule will trigger for every TLS packet which initiated from the internal network `$HOME_NET` to the external network, which is a TLS Handshake Client Hello message with the `Cipher Suites Length` field with the value 28 (14 cipher suites supported).

```
# This rule checks for ciphersuites_length in TLS ClientHello packet
# It does not alert, only sets a state for the next rule
alert tcp $HOME_NET any -> any any \
  (content:"|16|"; offset:0; depth:1; \
  content: "|00 1c|"; offset:44; depth:47; \
  flowbits:set,tor_browser; flowbits:noalert; sid:1000001)
```

Listing 3.1: Client Hello rule

Snort will not generate an alert for each packet that matches this rule. Instead, the `tor_browser` variable will be set with the `flowbits:set` rule option. The alert is only generated if the response packet matches one of the two rules following this one (that is the purpose of the rule option `flowbits:isset`). Listing 3.2 shows these two rules. Basically, they will match response (Server Hello) packets that have the `Extension Length` field with the values 13 or 18.

```
# These rules checks for extensions_length in TLS ServerHello packet
# It only alerts if the previous have saved the state
alert tcp any any -> $HOME_NET any \
  (msg: "TOR BROWSER DETECTED ! ! !"; \
        content:"|16|"; offset:0; depth:1; \
  content:"|00 0d|"; offset:47; depth:50; \
  flowbits:isset,tor_browser; sid:1000002)
alert tcp any any -> $HOME_NET any \
  (msg: "TOR BROWSER DETECTED ! ! !"; \
        content:"|16|"; offset:0; depth:1; \
  content:"|00 12|"; offset:47; depth:50; \
  flowbits:isset,tor_browser; sid:1000003)
```

Listing 3.2: Server Hello rule

Note that the values expressed here correspond to hexadecimal values in the rules. Also, the assumption that only Client Hello or Server Hello messages can match the rules can be violated, because the protocol following TCP can be other than TLS. However, it is unlikely that a different protocol exactly matches the `content` rule options specified.

Further, Snort can recognize TLS sessions with the `ssl_state` rule option, when the SSL preprocessor is enabled. The usage of `ssl_state:client_hello` and `ssl_state:server _hello` could be used to replace the `content:"|16|"`, and prevent the previous unlikely problem. Nonetheless, using that rule option did not produce the expected results, hence the previous adaptation.

Because the administrative actions after a violation are context-dependent, they are not covered here. For example, the actions of adding the IPv4 address that generated the violation to an *Access Control List (ACL)* or dropping the packet may not be intended.

## 3.4 SUMMARY

This chapter covered all the details and assumptions about the test case employed. It starts by examining the observer's position and the portion of traffic that it can monitor. This work assumes that the observer position lies in the middle of the onion proxy and the guard node. Then, Tor TLS traffic characteristics are discussed, aiming at the identification of variables of interest to create the proper rules. The chapter ends with an explanation of the rules created to detect Tor related traffic.

# 4

# EXPERIMENTAL EVALUATION

This chapter starts with a brief introduction about the test environment and a description about the evaluation process. As will be described, two different tests were performed. Then, the results of the two tests are shown and analyzed in detail. The chapter ends with a discussion about the results.

## 4.1 TEST ENVIRONMENT

To test the rules presented so far, background traffic was generated, along with Tor connections. Two types of tests were carried out. The simpler consisted of traffic generated from the browsers studied in the previous chapter. The more complex test consisted of real traffic captured in an enterprise.

The Tor connections were originated from a script that executes the Tor Browser Bundle to connect to the Tor network 17 times, each time to a different guard node [1]. To prevent the case where the first connection is always established with the same guard node, the solution was to add each node IPv4 address to the list of wanted `EntryNodes` in the configuration file `torrc`. This node rotation makes the test more complete because it can detect cases where guard nodes use a different handshake. Only one address is present in the configuration file at a time, replacing each other from subsequent executions.

Both background traffic and Tor specific traffic were locally sniffed with `tcpdump`, and then merged to a final file. Then, Snort takes that file as input to test the rules created. The process is illustrated in Figure 4.1.

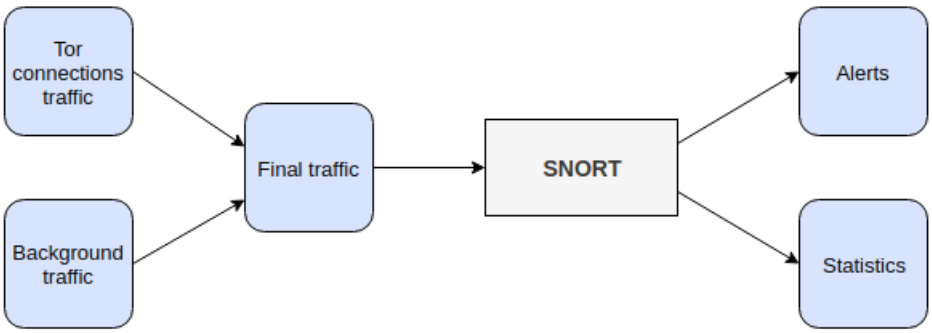



Figure 4.1: Test environment

1 50 guard nodes were used, but only 17 were successfully connected

### 4.1.1 *Browsers traffic*

In this first test, a python script has established connections to the fifty more accessed websites in Portugal[2], using the five web browsers discussed in subsection 3.2.3. The purpose of using different browsers is to have Client Hello messages varying the cryptographic parameters supported, and thus offered. In this way, at least 250 handshakes are performed (some websites perform more than one). The script was executed in a laptop within a domestic network.

### 4.1.2 *Real traffic*

In the second test, a `Sophos XG 105 Firewall` sniffed all traffic generated by 10 persons in the same *Local Area Network (LAN)* for one hour. The traffic passing through the firewall does only include each person's laptop in that LAN. The resulting network trace includes lots of different applications, such as *OpenVPN*, *CiscoVPN*, *Microsoft OneDrive*, *Microsoft Teams*, *Skype*, *Slack*, *Spotify*, *EMC Avamar*, *Tortoise SVN*, *NetBeans* or *Maven*. This is not a complete list of applications, and the trace could have more applications. Also, those 10 persons were not alerted to prevent interfering with the purpose of the capture.

## 4.2 EVALUATION

As previously said, Snort takes the merged files as input and logs the triggered alerts to a file. Simultaneously, Snort displays some statistics about the traffic evaluated, such as the time execution, the total numbers of packets, the number of alerts generated or information about TLS packets.

The accuracy of the rules created is measured by checking if the number of Snort alerts about Tor is the same as the connections established in the script. That is, both tests must give 17 alerts about Tor to achieve an accuracy of 100%. More alerts not related to Tor do not interfere with the results. Further, each false positive drowns a false negative (or vice-versa). For example, the obtained results could have 17 alerts about Tor, 16 true positives, one false positive and one false negative. To prevent this, each packet is compared with the file containing only Tor traffic. This approach avoids misunderstanding the results as a theoretical accuracy of 100% would be wrong.

## 4.3 RESULTS

A preliminary test showed that executing the Tor Browser generated an alert in Snort. Further refinements to the tests showed that it is possible to have false positives using the rules defined.

The first test, with traffic generated by five different browsers, had 100% accuracy. In a more realistic scenario, the second test has generated 70 alerts, but only 21 were triggered by the rules created. The other alerts were triggered by community rules already included in the configuration file. Because only 17 Tor connections were established, this gives 4 false positives. These false positives were all related to the same service, namely the `Microsoft Teams` desktop application. Inspecting these 4 alerts proved that the TLS handshake parameters are exactly the same as the

2 https://www.similarweb.com/top-websites/portugal

rules created. The other 17 alerts about Tor were manually confirmed to actually belong to Tor, by comparing the IPv4 addresses in the logs to the addresses used in the script to force Tor connections purposely.

Figure 4.2 illustrates the number of total packets of each test case and the number of packets that Snort could analyze. As shown, Snort could analyze all packets. The real traffic test case was analyzed in less than 3 seconds, which gives approximately 8 Gbps of throughput. However, remember that the analysis was not in real-time. In Figure 4.3, the number of Tor alerts and logs are shown. Snort could correctly log all the alerts generated, but the real traffic network trace gave 4 more Tor alerts.

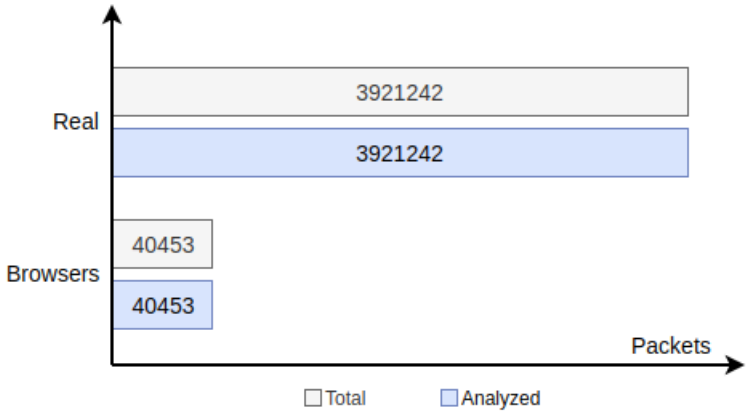



Figure 4.2: Number of packets per test

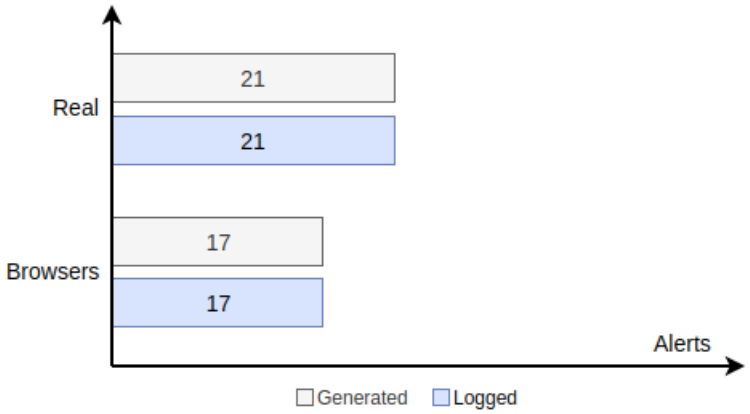



Figure 4.3: Number of Tor alerts per test

In subsection 3.2.3, it was discussed the importance of the `session ID` field. The rules created have each specific offset because when analyzing Tor traffic, no session identifiers were present. When there is no session identifier, the `Client Hello` message has only one field in this context, the `Session ID length` with the value zero. When there is an identifier, the `Client Hello` message has the length field and the corresponding value following it. Hence, when creating the rules, the field following the `session ID length` was not the actual value, but the next, which is the `cipher`

`suites length` field. All the 4 instances that were erroneously classified had a session identifier, which should make them not trigger the alerts. However, the alerts were generated, so it is assumed that Snort can ignore the identifier value or shift the offset accordingly.

| | **Browser** | **Real** |
|---|---|---|
| Client Hello | 1626 | 13567 |
| Server Hello | 578 | 13485 |
| Certificate | 420 | 4579 |
| Server Done | 2166 | 26393 |
| Client Key Exchange | 1164 | 4733 |
| Server Key Exchange | 71 | 2165 |
| Change Cipher | 2592 | 26060 |
| Client Application | 1674 | 15063 |
| Server Application | 720 | 8146 |
| Alert | 79 | 529 |
| **TLS packets** | **8760** | **159467** |

Table 4.1: TLS statistics

The values in Table 4.1 refer to specific statistics about the TLS traffic in both tests. The test with browsers has a bigger percentage of TLS traffic among its total traffic. This is because the captured trace was about requesting each website only via HTTPS, which gives this higher amount of TLS traffic. The real traffic test has a higher diversity of applications and protocols, which leads to a decrease in the percentage of TLS traffic.

## 4.4 DISCUSSION

The first test has generated 17 alerts, all about Tor. By examining all the IPv4 addresses in the logs, it was confirmed that all addresses were Tor nodes with no duplicates. This result is not surprising because the study of the TLS parameters used by browsers was previously made. However, the requests made with the browsers had the same version used in that previous study. The results could have been different if the requests were made to the same websites with different browsers' versions [22].

Note that the requests to the 50 different websites are only meaningful if the length of the `cipher suites` offered in the Client Hello equals that of Tor. To the same version, browsers always offer the same set of cipher suites. If the first set differs, the rules will not trigger an alert (at least Tor alerts), and the remaining 49 will differ too. In case that the set is equal, the results will depend on the website's behavior when handling requests.

The false positives encountered in the second test originated from the same service. Like this one, more services can match rules' parameters because this work only tried to distinguish the Tor Browser among other browsers. By previously enumerating more services, the need to introduce more parameters to the rules would arise beforehand. For example, if the `Microsoft Teams` appli-

cation that generated the false positives was included in the study (like the browsers), one more rule could be added to check the parameters of the server's certificate. Optionally, other parameters could be included in the rules created. For example, the SNI extension in the Client Hello could be examined and compared to a list of trusted server names.

## 4.5 SUMMARY

The results have shown that the rules created can correctly identify Tor traffic. More specifically, can correctly identify that a Tor-like TLS handshake was performed. Some services can match the chosen parameters, as the second test showed. The possible outcomes to this are to analyze each alert and check if its IPv4 address is related to some known service, if its SNI extension is not a random string, or if the certificate is not from a known service.

# 5

# CONCLUSION

This work shows that Tor can be detected. Ensuring that Tor is undetectable increases its security by bringing more users to the network. As previously stated, usability is a security requirement in areas such as traffic analysis. Tor traffic should be better disguised because its usage is problematic in some censorship countries. The rest of the conclusion revisits the research questions and summarizes the findings.

*What are the characteristics of Tor traffic in terms of traffic analysis?*
The characteristics of Tor traffic can be identified in traffic analysis. In the one hand, specific IPv4 addresses are publicly related to Tor nodes. On the other hand, the TLS handshake is a plaintext process that leaks information about Tor usage.

*How does Tor provide online anonymity and how is it different from other traffic?*
Tor only provides online anonymity if no personal information is submitted. Assuming this, it provides online anonymity by bouncing the traffic through more nodes than what would be necessary. By doing this, it guarantees that each node can only know one thing from two possible. Either the service being accessed, or who is accessing it.

*Are there any changes that could make Tor look like common traffic?*
Yes. Tor should upgrade its TLS version in the first place to leak less information in the TLS handshake process. Also, it should better mimic its traffic with browser's traffic by continuously updating it accordingly.

*Can Tor traffic be effectively detected?*
It depends. Tor traffic between an onion proxy and the guard node, or between two onion relays can be detected, if one can monitor the TLS handshake. Tor detection without looking at the handshake was not covered in this work (that includes Tor traffic between exit relays and websites which do not perform a Tor-like handshake).

Knowing that Tor is detectable, performing some tests showed that the number of false positives depends on the type of traffic, and their probability tends to increase as the number of different applications used in the infrastructure increases.

With respect to this area of research, it will remain an arms race between the entities that search for online anonymity and the ones that search for surveillance/censorship. The investigation showed

that currently there is no one-fits-all solution to this problem. Different strategies have arisen due to the increasing interest in this area, but those only solve the problem to some extent.

## 5.1 FUTURE WORK

Throughout this work, it was assumed that clients were not able to modify Tor's built-in options, files' configuration, or any library that could change the normal operation of Tor. In the real world, the rules presented here would be useless if a client could effectively change its normal behavior to overcome the rules created. Because of that, more research must be done to address these challenges. For example, by changing the `openssl` library to offer fewer cipher suites than the ones commonly used by Tor, a client could surpass the detection approach.

In addiction, continuous research is important because a software update can completely overcome the rules created before that update (unless an intelligent or autonomous system can adapt to those changes). Another line of investigation is to study not only the observer's position studied here but also between two onion proxies and between the exit node and the final destination. TPT is mainly used between two onion proxies. As so, it is a topic that should be better covered in future works.

Most importantly, the anonymous systems research line must continue to evolve to allow everyone to have online privacy. This includes Tor, which has to invest resources to continue the arms race against all the privacy intruders. Fortunately, many works come from academic and nun-funded research, because it is an open-source project, non-profitable, and a topic that is important to the common good.

# BIBLIOGRAPHY


[1] Stephanie Winkler and Sherali Zeadally. An analysis of Tools for Online Anonymity. *International Journal of Pervasive Computing and Communications*, 11(4):436–453, 2015.

[2] Kishan Neupane, Rami Haddad, and Lei Chen. Next Generation Firewall for Network Security: A Survey. *Conference Proceedings - IEEE SOUTHEASTCON*, 2018-April:1–6, 2018.

[3] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean Alexander Müller, and Klaus Hänßgen. A Survey of Payload-based Traffic Classification Approaches. *IEEE Communications Surveys and Tutorials*, 16(2):1135–1156, 2014.

[4] Alberto Dainotti, Antonio Pescapé, and Kimberly C. Claffy. Issues and Future Directions in Traffic Classification. *IEEE Network*, 26(1):35–40, 2012.

[5] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, 6th edition, 2012.

[6] Sasan Adibi. Traffic Classification Packet- , Flow- , and Application-based Approaches. 1:6–15, 2010.

[7] Taner Yildirim and PJ Radcliffe. Voip traffic classification in ipsec tunnels. In *2010 International Conference on Electronics and Information Engineering*, volume 1, pages V1–151. IEEE, 2010.

[8] Ramzi A. Haraty and Bassam Zantout. The TOR Data Communication System: A Survey. *Proceedings - International Symposium on Computers and Communications*, Workshops:1–6, 2014.

[9] Roger Dingledine and Nick Mathewson. Design of a Blocking-resistant Anonymity System. *Svn.Torproject.Org*, pages 1–24, 2006.

[10] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[11] Anders Olaus Granerud. Identifying TLS Abnormalities in Tor. *Information Security*, 2010.

[12] V. I. Pistunovich. Tor: The Second-Generation Onion Router. *Soviet Atomic Energy*, 46(4):337–337, 2005.

[13] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. 2014.

[14] Low-resource Routing Attacks Against Tor. page 11, 2007.

[15] Heenan Ross and Moradpoor Naghmeh. A Survey of Intrusion Detection System Technologies. *The First Post Graduate Cyber Security Symposium The Cyber Academy, Edinburgh Napier University*, (May):317–322, 2016.

[16] Martin Roesch. Snort Users Manual 2.9.13. 2019.

[17] Antonio Montieri, Domenico Ciuonzo, Senior Member, and Giuseppe Aceto. Anonymity Services Tor , I2P , JonDonym : Classifying in the Dark (Web). (February), 2018.

[18] Robert Koch, Mario Golling, and Gabi Dreo Rodosek. How Anonymous Is the Tor Network? A Long-Term Black-Box Investigation. *Computer*, 49(3):42–49, 2016.

[19] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Fair and Accountable Anonymity for the Tor Network. 4(Icete):560–565, 2017.

[20] Jeremy A. Stone, Neetesh Saxena, and Huseyin Dogan. Systematic Analysis: Resistance to Traffic Analysis Attacks in Tor System for Critical Infrastructures. *Proceedings - 2018 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018*, pages 2832–2837, 2019.

[21] Johanna Amann and Robin Sommer. Exploring Tor's Activity Through Long-term Passive TLS Traffic Measurement.

[22] Martin Husák, Milan Čermák, Tomáš Jirsík, and Pavel Čeleda. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *Eurasip Journal on Information Security*, 2016(1):1–14, 2016.