Universidade de Aveiro
2022

Bruno Filipe
Oliveira Aguiar

Deteção de atividades ilícitas de software Bots através do DNS

Detection of illicit software Bot activities over DNS

Universidade de Aveiro
2022

Bruno Filipe Oliveira Aguiar

Deteção de atividades ilícitas de software Bots através do DNS

Detection of illicit software Bot activities over DNS

"*Human beings are born with different capacities. If they are free, they are not equal. And if they are equal, they are not free.*"

— Aleksandr Solzhenitsyn

Universidade de Aveiro
2022

Bruno Filipe Oliveira Aguiar

Deteção de atividades ilícitas de software Bots através do DNS

Detection of illicit software Bot activities over DNS

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Paulo Jorge Salvador Serra Ferreira, Professor associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Pétia Georgieva Georgieva, Professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

agradecimentos / acknowledgements

Em primeiro lugar, gostaria de agradecer à minha família (Andreia Inlcuída) e amigos o apoio que me têm dado ao longo de todo o meu percurso académico, e em especial aos meus pais e avós onde sempre me tentaram dar todas as condições necessárias para que eu pudesse ter aqui chegado.
Gostaria também de agradecer aos meus orientadores, Prof. Paulo Salvador e Prof. Pétia Georgieva, por estarem sempre disponíveis em me ajudar, por tudo o que me transmitiram e por todas as nossas trocas de ideias.
Por último mas não menos importante, gostaria de fazer um agradecimento especial ao Éder, o membro de quatro patas da família e de uma lealdade (e energia) imensa, que esteve sempre ao meu lado, muitas vezes no sentido literal, durante a maior parte desta jornada.

Palavras Chave

DNS, Deteção de Malware Bots, Detecção de Botnets, Detecção de Novidades, Aprendizagem Automática

Resumo

O DNS é um componente crítico da Internet, já que quase todas as aplicações e organizações que a usam dependem dele para funcionar. A sua privação pode deixá-las de fazerem parte da Internet, e por causa disso, o DNS é normalmente o único protocolo permitido quando o acesso à Internet está restrito. A exposição constante deste protocolo a entidades externas obrigam corporações a estarem sempre atentas a software externo ilícito que pode fazer uso indevido do DNS para estabelecer canais secretos e realizar várias atividades ilícitas, como comando e controlo e exfiltração de dados.
A maioria das soluções atuais para detecção de malware bots e de botnets são baseadas em técnicas inspeção profunda de pacotes, como analizar payloads de pedidos de DNS, que podem revelar informação privada e sensitiva. Além disso, a maioria das soluções existentes não consideram o uso lícito e cifrado de tráfego DNS, onde técnicas como inspeção profunda de pacotes são impossíveis de serem usadas.
Esta dissertação propõe mecanismos para detectar comportamentos de malware bots e botnets que usam o DNS, que são robustos ao tráfego DNS cifrado e que garantem a privacidade das entidades envolvidas ao analizar, em vez disso, os padrões comportamentais das comunicações DNS usando estatística descritiva em métricas recolhidas na rede, como taxas de pacotes, o tamanho dos pacotes, e os tempos de atividade e silêncio. Após a caracterização dos comportamentos do tráfego DNS, um estudo sobre os dados processados é realizado, sendo depois usados para treinar os modelos de Detecção de Novidades.
Os modelos são treinados com dados lícitos recolhidos de multiplas atividades lícitas, como ler as notícias, estudar, e usar redes sociais, em multiplos sistemas operativos e com multiplas configurações. De seguida, os modelos são testados com dados lícitos semelhantes, mas contendo também tráfego de malware bots. Os nossos testes mostram que com modelos de Detecção de Novidades é possível obter taxas de detecção na ordem dos 99%, e de 98% para malware bots que geram pouco tráfego.
Este trabalho finaliza com algumas ideas para uma geração de tráfego ilícito mais realista, já que as ferramentas atuais de DNS tunneling são limitadas quando usadas para imitar usos de DNS lícito, e para uma melhor deteção de situações onde malware bots geram pouco tráfego.

Keywords DNS, Bot Malware Detection, Botnet Detection, Anomaly Detection, Novelty Detection, Machine Learning

Abstract

DNS is a critical component of the Internet where almost all Internet applications and organizations rely on. Its shutdown can deprive them from being part of the Internet, and hence, DNS is usually the only protocol to be allowed when Internet access is firewalled. The constant exposure of this protocol to external entities force corporations to always be observant of external rogue software that may misuse the DNS to establish covert channels and perform multiple illicit activities, such as command and control and data exfiltration.

Most current solutions for bot malware and botnet detection are based on Deep Packet Inspection techniques, such as analyzing DNS query payloads, which may reveal private and sensitive information. In addiction, the majority of existing solutions do not consider the usage of licit and encrypted DNS traffic, where Deep Packet Inspection techniques are impossible to be used.

This dissertation proposes mechanisms to detect malware bots and botnet behaviors on DNS traffic that are robust to encrypted DNS traffic and that ensure the privacy of the involved entities by analyzing instead the behavioral patterns of DNS communications using descriptive statistics over collected network metrics such as packet rates, packet lengths, and silence and activity periods. After characterizing DNS traffic behaviors, a study of the processed data is conducted, followed by the training of Novelty Detection algorithms with the processed data.

Models are trained with licit data gathered from multiple licit activities, such as reading the news, studying, and using social networks, in multiple operating systems, browsers, and configurations. Then, the models were tested with similar data, but containing bot malware traffic. Our tests show that our best performing models achieve detection rates in the order of 99%, and 92% for malware bots using low throughput rates.

This work ends with some ideas for a more realistic generation of bot malware traffic, as the current DNS Tunneling tools are limited when mimicking licit DNS usages, and for a better detection of malware bots that use low throughput rates.

Contents

List of Figures

List of Tables

Acronyms

DDoS	Distributed Denial-of-Service
C&C	Command and Control
IP	Internet Protocol
P2P	Peer-to-Peer
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
DPI	Deep Packet Inspection
DGA	Domain Generation Algorithm
IRC	Internet Chat Relay
TLD	Top-Level Domain
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
RFC	Request for Comments
RR	Resource Record
GDPR	General Data Protection Regulation
PII	Personal Identifiable Identification
TLS	Transport Layer Security
DoT	DNS over TLS
DoH	DNS over HTTP
GNS3	Graphical Network Simulator-3
SSH	Secure Shell Protocol
NAT	Network Address Translation
EDNS	Extension Mechanisms for DNS
pps	Packets per second
MI	Mutual Information
NP	Nondeterministic Polynomial time
SFS	Sequential Forward Selection
SBS	Sequential Backward Selection
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
RFE	Recursive Feature Elimination
SVM	Support Vector Machine
PCA	Principal Component Analysis
PC	Principal Component
AD	Anomaly Detection
pdf	Probability Density Function
MLE	Maximum Likelihood Estimation
FPR	False Positive Rate
EM	Expectation-Minimization
KDE	Kernel Density Estimation
kNN	k-Nearest Neighbors
LOF	Local Outlier Factor
SSL	Secure Sockets Layer
API	Application Programming Interface
ADS	Anomaly Detection Server
GMM	Gaussian Mixture Models
RAM	Random-access memory

CHAPTER 1

Introduction

This chapter describes the motivations that led up to the development of this Dissertation, since the misuse of DNS to set up covert channels for malevolent purposes, to the role of Machine Learning in Network-based detection mechanisms. We also address the main contributions of this Dissertation, as well as a summary of its structure.

1.1 Motivation

The Internet has seen its use to grow steeply in the last decades with the digital transformation of business processes, services and even reshaping cultures and interpersonal relationships. With a wide heterogeneously of end devices communicating with each other over wired, wireless, or cellular networks, corporations need to be in a constant alert state as networks face nowadays multiple threats that are able to bypass traditional security mechanisms and exploit inherent vulnerable systems.

Network devices are frequently compromised by external rogue software programmed to perform multiple illicit activities. That rogue software agents are called as malware bots (or illicit software bots) and they try to establish a covert channel with a rogue entity that coordinates and manages the attacks. The communication between the bot and its external control is usually called Command and Control (C&C), and can be performed with stealth channels. One of the most common ways to archive this state of stealthiness is the usage of DNS to create an hidden data channel, as DNS is an important component of all Internet applications and organizations since it allows them to be part of the Internet, and so its access needs to be always granted. The Figure 1.1 depicts an example of a illicit usage of DNS that exfiltrates a password.

Moreover, threat actors may not employ only one bot, but a illicit network of infected devices within organizations to more efficiently execute their illicit activities. This illicit network is called a botnet and they can be organized in centralised or distributed

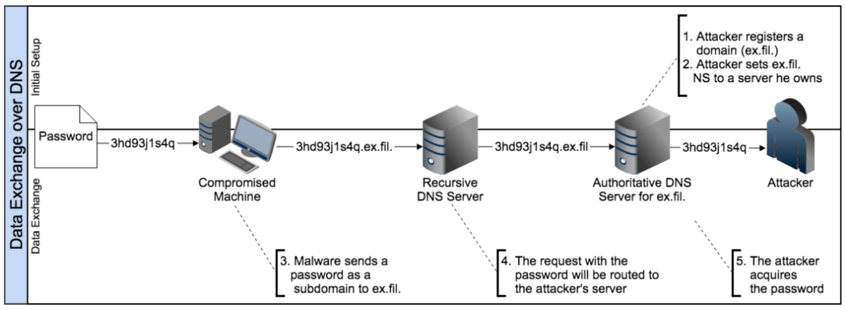

Data Exchange over DNS
Initial Setup
Data Exchange
1. Attacker registers a domain (ex.fil.)
2. Attacker sets ex.fil. NS to a server he owns
Password
3hd93j1s4q
3hd93j1s4q.ex.fil.
3hd93j1s4q.ex.fil
3hd93j1s4q
Compromised Machine
Recursive DNS Server
Authoritative DNS Server for ex.fil.
Attacker
3. Malware sends a password as a subdomain to ex.fil.
4. The request with the password will be routed to the attacker's server
5. The attacker acquires the password

Figure 1.1: Misuse of DNS for data exfiltration purposes. Source: Detection of malicious and low throughput data exfiltration over the DNS protocol [1].

architectures, were each bot can be have different roles in the botnet, for instance as a bot that relays commands to other bots, a bot that executes commands, or a data exfiltration bot.

The prevention and detection of such C&C channels is complex because DNS is a common and vital service to network and services operations, and illicit activity is deeply embedded in it and it is hardly distinguishable from licit uses of the service. Botmasters often create little traffic, or else their detection can led to persecution and detention, therefore, the botnet traffic is generated, most of the time, on the infected end-devices, and the more automated the bot end is, the less traffic it generates.

The detection of these anomalous behaviors must rely on constant monitoring of the DNS services and network activity. With the advent of new and sophisticated botnets, security professionals noticed that relying in approaches that only detect based on patterns and behaviors of well-known botnets is not enough, forcing them to move to methodologies that jointly use statistical and machine-learning techniques for the identification of new, unseend phenomena.

Service anomalous behaviors can be detected using a Host-based approach, or using a Network-based approach. The former monitors the internal behavior of computer systems and analyses their process activities such as log files and system call arguments, while the latter detect the illicit usage of resources by solely monitoring network traffic behaviors over time activities. Host-based techniques can be implemented in terminal devices, or in DNS Servers, however, most Machine Learning models are computationally expensive to run in end devices and externally hosted DNS Services may impose restricted management and control over them. By monitoring at a network level, one is not restricted with the host's computational resources and restrictions, that might limit, not only the efficiency of Machine Learning algorithms, but also the performance capabilities of the host itself.

With the traction that the Internet gained over the last years, privacy concerns about the processing of identifiable and sensitive data were also on the rise. Companies and organizations must meet their legal obligations accordingly to data protection and privacy legislations when collecting data that can, directly or indirectly, identify the data subjects. Under such regulations, network approaches based on Deep Packet Inspection (DPI) techniques are often considered invasive as they look at the content of packets to process such important data for botnet detection, therefore one should use as minimal personal information as possible. On top of that, more and more services are switching to secure and encrypted DNS, such as DNS over HTTPS (DoH) and DNS over TLS (DoT) with the objective to prevent eavesdropping and the manipulation of DNS payloads, hence using DPI techniques are simply not feasible in those scenarios.

1.2 Objectives

The main goal of this Dissertation is to provide a robust and data privacy focused solution that detects if a device has been infected with a malware bot under a corporate network. The large majority of current solutions are invasive and often deal with DPI techniques that might contain actual data, for instance, the examination of DNS payloads, such as using information entropy on DNS queries and the identification of TCP and UDP packet headers. Naturally, this collection of personally identifiable data also rise some privacy concerns, that were discussed early on, in Section Motivation. In the contemplation of data privacy laws, our solution only uses IP addresses to distinguish between the device being monitored and the DNS server, important for the collection of upstream and downstream metrics.

Moreover, the usage of the DNS protocol by malware bots has been largely overlooked in the literature, specially when the topic is low throughput malware communications and the advent of licit and illicit DNS traffic over encrypted protocols. Detecting encrypted DNS communications can be challenging, for the reason that DNS traffic can be blended with traffic from other applications that also use the same encrypted protocols. Nevertheless, companies may impose the usage of their own enterprise-operated DNS servers, or the usage of allowed externally hosted services, blocking the other third-party alternatives. This way, companies can ensure if the encrypted traffic is DNS traffic or not. Additionally, licit encrypted DNS traffic most often shows very different patterns from non-encrypted licit DNS usages, and Machine Learning models should be able to not wrongly flag that traffic as malicious.

For the aforedescribed reasons, the proposed solution also aims to detect even the *slowest and lowest* throughput malicious communications over DNS under encrypted and non-encrypted DNS traffic, and this approach must stop well-known botnet attacks as well as zero-day attacks as it relies in detecting anomalous patterns using Novelty Detection mechanisms, rather than focusing in well-known botnet behaviors.

The detection of new and unseen behaviors can be performed with semi-supervised algorithms that only train with unlabelled normal instances, constructing an accurate and meaningful representation of a licit usage of DNS. That model must be tested with labeled licit and illicit instances, classifying everything that deviates from the normal behavior as an anomalous data point.

1.3 Contributions

In short, the developed work can be summarised into four points:

- The proposal of privacy-focused Anomaly Detection (AD) systems for the detection of malware bots, using only IP addresses as personal identifiable data.
- Describe the processed metrics from network traces in a meaningful way with the objective to properly feed AD systems, increasing their convergence and performance.
- Accurately model a legitimate DNS behavior and detect a wide variety of possible illicit activities over DNS, using the existing semi-supervised Novelty Detection approaches.
- Validate the proposed approaches by designing multiple scenarios that emulate possible malware-generated traffic behaviors with the objective to precisely detect the presence of illicit communications in DNS.

1.4 Outline

The remainder of this Document is organized as follows:

1. **State of the art (Chapter 2):** Presents the context surrounding this Dissertation, since the evading mechanisms of malware bots and botnets, to their organization, and the current works regarding botnet detection techniques, specially the Machine Learning techniques currently used for AD.
2. **Methodologies for AD (Chapter 3):** Description of the design phase and the proposed mechanisms for the detection of anomalous DNS usages. It depics the entire pipeline, starting with the characterization of the scenario to be studied, the extraction of meaningful features over observation windows, the pre-processing of the created meaningful features, such as their normalization and dimensionality reduction, and how to properly train the Novelty Detection models.
3. **Results (Chapter 4):** Showcases multiple scenarios of malware communications using the DNS to be tested and the challenges associated with the emulation of those scenarios, presenting later the results obtained from modelling and testing the Novelty Detection algorithms.

4. **Appendices:** Depicts different graphs representing metrics derived from the raw data and features extracted from those metrics with the objective to give the reader a better visual understanding of the data.

CHAPTER 2

State of the Art

This chapter presents a brief background about malware bots and botnet communications, and the current state of the art relating to their detection, being divided into four parts:

The first part is related to malware bots and botnets in general, more explicitly, about how malware bots communicate with external entities using C&C channels, about how they can infect other machines using active and passive propagation mechanisms to form a botnet, about how botnets can be organized, for instance, in centralized and decentralized topologies, and about how they can evade detection using common Internet protocols, such as the DNS.

The second part gives an overview of how DNS works, since the translation of host names into IP addresses, and *vice-versa*, to the format of DNS messages. The latter is specially important to understand since malware bots that use the DNS protocol need to comply with DNS norms and restrictions so that a successfully misuse of DNS can happen. Furthermore, this second part also explains how malware can encode data in DNS payload fields, and defines possible scenarios for the use of DNS as a covert channel for C&C communications and data exfiltration tasks.

The third part focuses on the current botnet detection solutions, explaining why relying in signatures of known botnets and Host-based solutions are infeasible. Moreover, it explains the advantages of using Network-based approaches to monitor a wide range of heterogeneous devices and why they are robust to code obfuscation and information concealing techniques.

Finally, the fourth section describes a series of Machine Learning techniques for AD, particularly feature engineering techniques, such as Feature Selection and Dimensionality Reduction techniques, and Novelty Detection algorithms that are robust to the presence of new and unseen phenomena. Many of these techniques will be used for AD of bot malware and botnet detection over DNS, as described in the following chapters.

2.1 Malware bots and botnets

Malware bots are a major cyber-threat for today's Internet, partially because they can effectively disrupt targeted Infrastructures, but also because malicious actors can simply rent malware services from cybercriminals without much effort, resulting in large economic damages for corporations and individuals [2], [3]. This rogue software is designed to perform a multitude of illicit activities [4], such as DDoS attacks, data exfiltration, malware dissemination, bitcoin mining, identity theft, and so forth, while taking control over the device's operations. The term "Bot" derives from the word "Robot", which refers to a physical or a virtual agent capable of following a set of actions, or instructions, in an automated way, mimicking, or replacing, the normal behavior of a human.

Being a bot by itself is not malicious. Because of their automated nature, bots can be used for licit and helpful purposes, such as performing customer services or indexing content for search engines, operating much faster than humans. However, like many other software applications, bots can also be used to perform malicious purposes. The Malicious bot may not be fully automated, but have an external entity controlling it. This entity is called the botmaster, and a Command and Control (C&C) channel can be established with his bot, allowing the botmaster to send instructions (commands) and receive data from the infected machine, and being an important communication infrastructure to coordinate, manage or suspend attacks [5].

Since attacking enterprise networks can be very profitable, cybercriminals often use sophisticated techniques to target specific companies and organizations. Instead of using a single bot to carry out malicious activities, malware programs can infect other terminal devices in the network, forming a distributed network of compromised devices. This network is called a Botnet, and they are a preferred method of attack due to the ease of computational and storage resource scalability and availability, their robustness to shutdowns, and because bots can be physically away from each other, in different timezones, countries, and thus under different laws, making them difficult to track.

The botmaster himself can use a number of infected devices, called stepping-stones [6], configuring them as proxy machines, SSH servers, or any other network redirection services [7], until making it to its C&C servers. In reality, C&C channels aren't necessarily directly linked to the botmaster, but to one or more C&C servers that act like rendezvous points, used by the botmaster to disseminate commands into the network, following the same philosophy of the bot end: they can be far away from each other, under different laws, making the bot network harder to take down. The

Figure 2.1 illustrates how a botmaster (Attacker) can access the C&C Servers through stepping-stones and control the botnet to carry out various illicit activities.

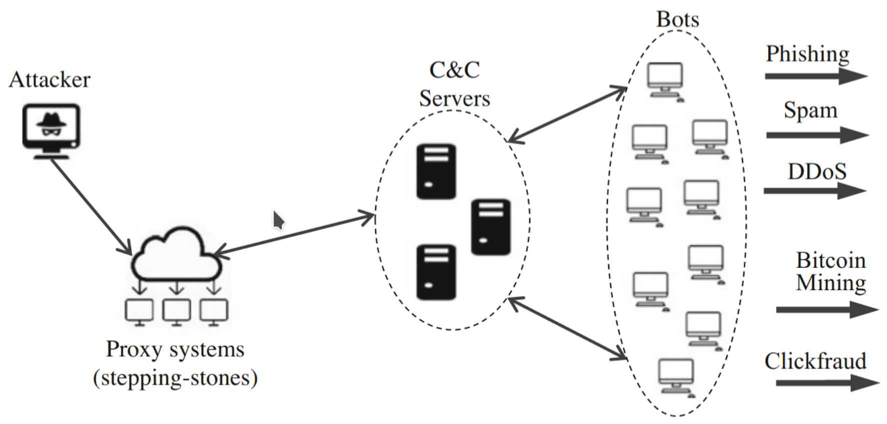


Figure 2.1: Typical Botnet structure. Source: Security of cyber-physical systems [6]

2.1.1 Introduction to the malware bot's life-cycle

In short, the life-cycle of a malware bot starts when the malicious program, in a binary form called *bot binary*, is executed on a given machine. Beforehand, malicious actors can choose different Propagation vectors in order to spread out the bot binaries. Once in the victim's machine, the bot agent may now initiate a process to discover its C&C servers and establish a covert C&C channel, forming an overlay network. This process is commonly called as *Rallying*. In an attempt to form a botnet, the infection can be spread to other machines using the same Propagating mechanisms that the malware bot used to get into the current infected host, or use new ones. To ensure that the underlying hosts remain infected and available for the longest period of time possible, and the botmaster and his botnet are not easily detected, the botnet needs to be designed to be as stealthy as possible, and so, different evasion mechanisms need to be developed.

2.1.2 Propagation vectors

One of the primary objectives for botmasters to achieve is to infect as many machines as possible, and one way to accomplish this is by looking for vulnerable ones. The exploitation can take place by scanning for one or more vulnerabilities in the targeted machine, such as weak security policies [8] (weak passwords, for instance), or for some zero-day vulnerability, in a stealthy manner. The exploitation of vulnerabilities can help the malware bot to gain administrative access over the vulnerable host, and these methods are usually called an **active propagation** vector because the bots are actively searching for vulnerable hosts in the network [9].

Another way of propagation is when the infection requires some form of user interaction. This is called **passive propagation** and can be done by making the use of social engineering methods, tricking the victim to install a certain malware, or to reveal confidential information. These means are built around human weaknesses and lack of knowledge of good security practices, and can be specially crafted to target specific victims of a company. Some examples of social engineering techniques are [10]:

1. **Phishing schemes:** where the victims are flooded with messages specially made to stimulate a sense of curiosity, urgency or fear on them.
2. **Pretexting:** where the attacker creates a real or invented scenario (pretext) impersonating a cow-worker, an authority agent, or someone that the victims will trust to reveal some form of sensitive information.
3. **Baiting/Trojan Horse:** fraudulent websites, or infected physical media, like USB pen sticks, that are made to look legitimate, guiding the victim to download and/or install the malware, carrying the attack.

2.1.3 The Rallying mechanism

Rallying is a mechanism that bots use to discover their C&C servers and to establish a hidden channel in order to be part of the botnet. The rallying can be done by using IP addresses, or the domain name. IP addresses are usually hardcoded in the bot binary, making them vulnerable to reverse engineering attacks, revealing their C&C server IPs, and thus being easily blocked [9]. P2P networks, however, tend to use a different hardcoding mechanism: they rely on a peer list, which contains a list of an initial set of active peers with a limited size [11], and they do not reveal the list for other bots in the network. This way, if the bot binary is reverse engineered, only a limited number of peers are exposed.

The botmaster can also use the DNS to easily re-map the domain name entry to another IP addresse if the others are shutdown, with no additional updates on the bot end [9], [11]. However this is still a hardcoded way of connecting to the botnet, since the domain name is static, which can be easily block listed. A more robust approach is to generate domain names using a DGA that pre-calculates a seed. The seed is known by the botmaster and the bot end computes it using an algorithm embedded in the bot binary. The botmaster can now choose one of the many pseudo-generated domains to be registered for its C&C servers, forming the C&C channel [12] infraestructure.

2.1.4 Botnet topologies

As aforementioned, a botnet can be seen as an overlay network of infected devices, independent from the underlying physical network. Nonetheless, they can be grouped in different network topologies, designed to be as stealthy as possible, or to suit a particular need. They don't need to be organized in a centralized manner, but adopt other topologies such as a distributed, or a hybrid one.

2.1.4.1 Centralized topologies

In centralized topologies, all the commands and reports in the botnet are sent and received by C&C servers. It is a simple topology, where all the bots in the bot end rely on a single centralized C&C server. This simplicity provides a low latency and makes the life of the botmaster easier, having only a single point of control. However, a single point of control, also means a single point of failure, and if the C&C server is taken down, the botmaster loses complete access to the botnet. To overcome the single point of failure, the C&C server can be replicated making the botnet more resilient if one of the C&C servers is shut down, and if the C&C servers are distributed in some geographical order, optimizations in the communication for a location-related bot end are possible.

A even more robust centralized topology is the Hierarchical one, which adds one or more layers of C&C proxies. They are also infected machines who relay commands sent by the C&C servers to the bot end. This setup is particularly difficult for security researchers to estimate the overall size of the botnet, but has the disadvantage of having a greater latency as commands need to pass through multiple communication branches, or hops [13].

2.1.4.2 Decentralized topologies

With a decentralized structure, the bot management is either distributed among the multiple C&C servers, or there is not really a hierarchy between the nodes in the network, where each node can be dynamically a master or a slave, and act like a bot or a C&C server [9]. These topologies are well defined, and can be grouped in the following way [13]:

1. **Peer-to-Peer**: In this topology, any peer node can receive commands from its botmaster and broadcast them to its known peer nodes in the peer list, under different possible routes, making the routing unpredictable. This makes a very resilient network to attacks, but the routing unpredictability makes some peers to be exposed to high latency and one can enumerate other peers by monitoring the communications of a single bot, or revealing a peer-list by a crawling attack.
2. **Hybrid Peer-to-Peer botnet**: only bots with a static global IP Address are candidates for being in the peer lists, behaving like both bot clients and C&C servers (servant bots), as peers with static global IP addresses tend to be more active for longer periods of time, with better response times and greater computational capabilities.

The fundamental botnet topologies aforedescribed can be seen in Figure 2.2.

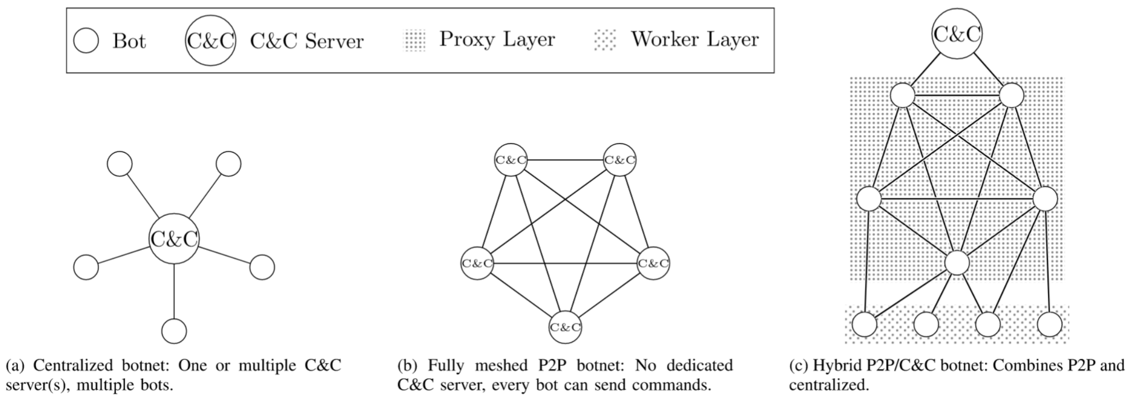

Bot
C&C
C&C Server
Proxy Layer
Worker Layer
C&C
(a) Centralized botnet: One or multiple C&C server(s), multiple bots.
(b) Fully meshed P2P botnet: No dedicated C&C server, every bot can send commands.
(c) Hybrid P2P/C&C botnet: Combines P2P and centralized.

Figure 2.2: Botnet topologies. Source: Botnet Communication Patterns [14]

2.1.4.3 More advanced topologies

Although the most common botnet topologies have already been listed, it is important to notice that novel variants from the above topologies can be developed, like, a stochastic peer-to-peer botnet, proposed by Ruitenbeek *et al.*[15], where each infected machine may exist in one of a set of possible stages, having different roles in the botnet, namely, as bot recruiter, or as a data exfiltration bot, being active or inactive, and being able to transit between stages, as one needs to perform a discrete event simulation to estimate effectively the size of the botnet over time.

2.1.5 Evading detection with covert channels and anonymization

For malware bots that rely on command and control in order to operate, covert channels play an important role as they are the backbone of the network, thus low latency and stealth communication mechanisms need to be implemented if one wants to evade from the deployed security measures. These mechanisms can be anything that enables the botmaster to send commands and the bot end to fetch them. It can be done by sending commands over the existing internet protocols, such as the HTTP protocol and the DNS protocol, or over applications, such as Social Networks, of which are not trivial to detect [16].

2.1.5.1 Common protocols for covert channels

One of the first protocols used for stealth communications was IRC protocol. IRC was a popular protocol for instant messaging, providing almost real time communication with the C&C servers. Despise that, its use in enterprise networks was not common, making it easy to distinguish from normal traffic [9].

A more viable protocol of evading in enterprise networks is to use the HTTP Protocol, as one can hide the bot malware traffic with normal web traffic, being easier to bypass firewall with port-based filtering mechanisms [17]. For instance, covert channels can be embedded in HTTP requests as encoded URL parameters, through HTTP cookies, or into the protocol header fields [18]. The HTTP protocol can also be used in P2P topologies, however, since it was originally designed for centralized topologies, replicated messages can loop throughout the botnet, thus additional workarounds need to be developed on top of the HTTP layer in order to prevent such application layer loops [14].

DNS, on the other hand, is a decentralized and vital protocol for companies and organizations, allowing them to be part of the Internet. Given its importance, it is not surprising that DNS is constantly under abuse by actors with malicious intentions, and its use in stealth communications will be discussed in more detail in the section DNS as a covert channel. The Figure 2.3 shows the most common application layer targeted protocols in cyber attacks, in 2015, where DNS is referenced as one of the most used.

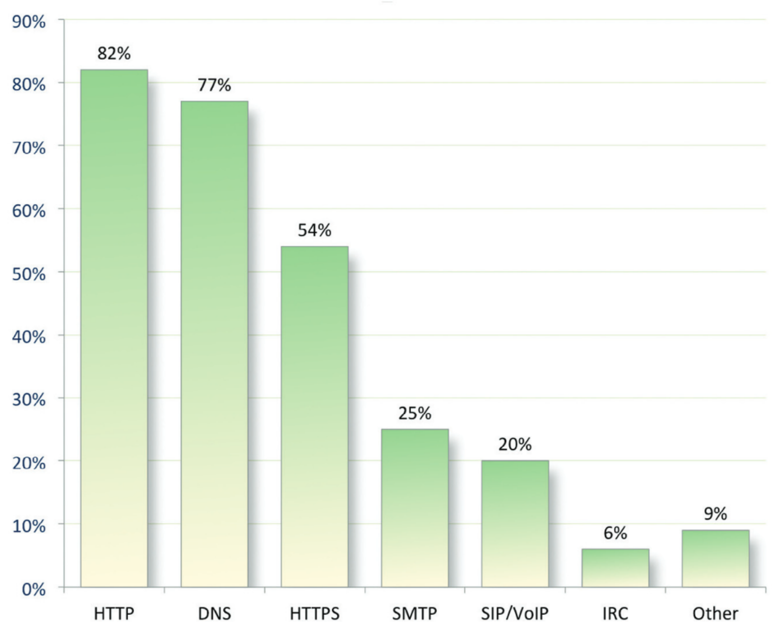


Figure 2.3: Common application layer targets (2015). Source: Arbor Networks

2.1.5.2 Information concealing

In an attempt to evade from DPI detection methods, botmasters have started to implement different information concealing techniques, such as encrypted covert channels, code obfuscation and stenography [16]. These techniques make content based analysis difficult and covert channels prevent the encrypted traffic from being discovered, as the information needs to comply with transmission norms and restrictions. Using encrypted methods, although, has some drawbacks. They might reveal abnormal communication patterns [18], increased throughput and larger message lengths that can be easily detected with statistical and temporal characteristics [9], when compared to unencrypted DNS traffic. Nonetheless, Internet services and applications are increasingly switching to secure and encrypted DNS communications, such as DNS over HTTPS (DoH) and DNS over TLS (DoT), with the intent to prevent eavesdropping, the manipulation of DNS payloads, and other man-in-the-middle attacks that might be menacing to the user privacy and safety. This traffic can be way more similar with the behaviors of encrypted C&C communications, thus their distinction may not be so trivial.

2.1.5.3 Anonymization of C&C Services

Further attempts in the anonymization of the botnet can be tried by the botmaster, that may also want to conceal the location of its C&C servers by using anonymized overlay networks, such as the Tor Network, hosting them as Tor hidden services [9]. Hidden services are services that are only accessible via Tor. When creating a hidden service, it is generated a hidden service descriptor containing its public key and the address of the Tor relays that it chose to be its introduction points. The clients (bot end) can access the hidden service using the public key encoded in the onion address, which downloads the descriptor, chooses a random Tor relay to act as a rendezvous point and inform the introduction points about its rendezvous address. In this manner, the C&C servers have their actual addresses and locations hidden, guaranteeing the service anonymity. Although in theory this may seem an interesting idea, in reality it suffers from the same vulnerabilities of the Tor Network. For centralized botnet architectures it is possible to discover the IP addresses of the hidden services by exploiting the use of the guard nodes, or performing traffic correlations between the bot end and the Tor Network nodes. In P2P architectures that create a low-throughput routing system to avoid traffic pattern analysis, crawling attacks can be performed against a given infected device, enumerating the bots in the network. Since one is not using dynamic IP addresses, but instead onion addresses, whose are uniquely assigned to each hidden service, this creates a serious threat to botnet's existence [19].

2.2 DNS as a covert channel

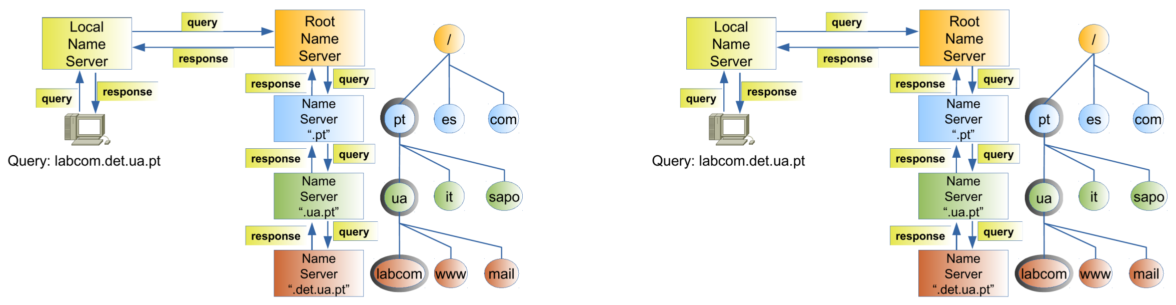
DNS is a mission-critical infrastructure where almost all Internet applications and organizations rely on, as they need to remain available all the time or else they disappear from the Internet. Even in heavily restricted corporate networks, DNS is one of the few protocols, if not the only one, that is not blocked by firewalls and its distributed design provides such advantages for botmasters over corporate networks [20], [21], as they can misuse it to exfiltrate data, send commands and tunnel other protocols.

2.2.1 DNS overview

As a globally distributed database system, DNS provides a mechanism of translation between host names and IP addresses, and *vice versa*, in a hierarchical manner, meaning that it resorts in root, TLD and authoritative name servers for the DNS resolution of a specific host name. Being an authoritative name server implies that it holds the DNS information for one zone, or else is just a local DNS cache serving client DNS queries [22]. The DNS root server is the name server who parses the TLD from the client's query, redirecting one's DNS resolver to a TLD name server based on its query extension (.com, .pt, .org, etc), whereas the TLD name server contains all the domain names that share a common domain extension [23]. They are both authoritative name servers for their specific zones [1], but one usually refers to authoritative name servers as name servers who provide definitive answers to DNS queries [24].

2.2.1.1 DNS Resolution

When a DNS resolution occurs, if a given domain is not stored in the local cache, the local name server starts querying other DNS servers in the hierarchy, having two different ways to do it so: (i) the Recursive way, where a DNS server queries other DNS server on behalf of original DNS client, and (ii) the Iterative way, where the local name server is forwarded to authoritative name servers, starting with the root servers [22]. In Figure 2.7 a schematic representation of Recursive and Iterative DNS can be seen.



(a) Recursive DNS **(b)** Iterative DNS

Figure 2.4: A graphical view of Recursive and Iterative DNS

2.2.1.2 DNS Transport

According to RFC 1035, one can transmit DNS messages in datagrams, using the UDP protocol, or in a byte stream, using the TCP protocol, on server port 53. Datagrams are the prefered method of sending queries due to their lower overhead and better performance, although for zone transfers one must use the TCP protocol because of

[1]https://www.iana.org/domains/root/servers

its reliability. In UDP, datagrams have a maximum size of 512 bytes, while in TCP connections large messages are break into multiple segments.

2.2.1.3 DNS Messages

DNS messages follow a single format (Figure 2.5), which is divided in 5 sections: Header, Question, Answer, Authority and Additional.

The Header is always present and specifies which of the remaining sections are present and need to be derived, e.g. if a message is a query or a response, a standard query or some other opcode, etc. The Header section has a static size of 12 bytes.

The Question section is used to carry a question to a name server, and it is composed by:

1. `QNAME`: contains a sequence of labels representing a domain name, where each label consists in a length octet followed by that that number of octets, terminating with the zero length octet for the null label of the root. The maximum length of a label is 63 bytes, or octets, due to the fact that the two most significant bits of the length octet are reserved for message compressing, whereas the maximum length of a query name is 255. As an example, the domain name `www.twitter.com` is encoded as follows:

3	'w'	'w'	'w'	7	't'	'w'	'i'	't'	't'	'e'	'r'	3	'c'	'o'	'm'	0

2. `QTYPE`: specifies the query type with a size of 2 bytes.
3. `QCLASS`: specifies the class of the query, with the same size of `QTYPE`.

The last three sections share the same format, namely a list of Resource Records. RRs are a collection of entries in a DNS zone database [25], and are used to specify information about a particular object, e.g. the DNS record type `A` is used to map a given domain name to a particular IPv4 address, while the DNS record type `CNAME` is used to map a given domain name to another domain name entry.

The Answer section contains a set of RRs that answer the question, the Authority section contains a set of RRs that point toward an authoritative name server, and the Additional section contains a set of RRs related to the query which are not strict answers for that question. Their set size are defined at the count field, in the Header section. When a given count field is zero, its corresponding section is empty.

The RR format is composed by:

1. `NAME`: the name of the owner to which the resource record pertains. Has a variable size.
2. `TYPE`: a 2-byte field containing one of the RR TYPE codes.
3. `CLASS`: a 2-byte field containing one of the RR CLASS codes.

4. `TTL`: a 4-byte signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached.
5. `RDLENGTH`: a 2-byte unsigned integer that specifies the length in octets of the `RDATA` field.
6. `RDATA`: a variable length string of octets that describes the resource.

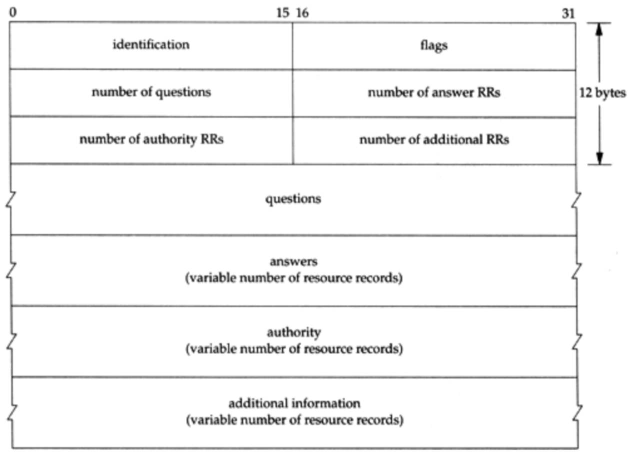


Figure 2.5: General format of DNS queries and responses. Source: TCP/IP illustrated (Vol. 1): The Protocols [26]

2.2.2 How DNS is misused for illicit activities

Usually, exfiltrated data and commands are encoded in the DNS payload fields (`QNAME` and `RDATA`) over a RR request [27]. Data can be encoded in the host name, and the botmaster can encode up to 63 characters in each label, where each character must be a letter, a number or a hyphen, and the characters must be all upper case or lower case, leaving one with an alphabet of 37 different possible characters. To effectively send data over DNS, encoding schemes need to comply these restrictions.

However less commonly used in licit DNS traffic, Resource Records that have fewer restrictions, like the `TXT` Record which is case-sensitive, can be employed, and so, encoding schemes that use less bits per character are possible. For instance, Base32 is a 5-bit encoding scheme that can be used in RRs with stricter restrictions, and Base64 in RRs similar to the `TXT` Record.

2.2.2.1 The DNS threat landscape

Asaf Nadler *et al* [1] defined two possible scenarios for the use of DNS as a covert channel: as low throughput data exfiltration and high throughput DNS tunneling. Each scenario has different communication patterns. DNS Tunneling software is typically more reliable, as it relies in frequent *keep-alive* messages to inform that the client software is alive and waiting for commands. These messages need to be different to each other in order to bypass caching services in the go, that may drop queries if they are identical to the recently received ones. DNS Tunneling software is used to establish bi-directional and interactive channels, and they usually produce *lengthy* messages, as packets with larger sizes decrease the likelihood of network congestion that, other way, could cause an increase in packet loss and network latency.

In low throughput exfiltration, malware exchange queries less frequently, in a more *opportunistic* and *unexpected* manner, and their packet sizes are better crafted as well, meaning greater evasion from traditional security measures. Their connection is typically unidirectional, receiving from the remote server, at most, acknowledge messages. These slow and stealthy attacks are planned to be performed over longs periods of time, like weeks or months, and are not usually detected via traditional defense systems.

DNS Tunneling software is typically used for C&C communications where *keep-alive* messages inform the botmaster about the status of his bots, and their low-latency nature enables a rapid dissemination of commands in the botnet. These features make the illicit traffic much easier to be detected, and so, advanced C&C communications may be implemented with more discrete and unexpected solutions. In data exfiltration tasks, malware bots may mimic the licit DNS traffic of the host machine, or generate less traffic in a low throughput manner, by blending with the licit traffic in order to evade detection.

2.3 Botnet Detection techniques

As aforementioned in Section 2.1, business need to keep ahead with the growing and novel botnet threats, or else they may see their security compromised and suffer massive economic losses, therefore, botnet detection mechanisms need to be deployed. Detection of malware bots and botnets can be mainly grouped in three categories [22]:

1. **Honeypot-based detection**: computer systems which don't produce any value are used as honeypots with the intention to lure malware from its actual target [28]. This technique is useful to obtain malicious code samples, and analysis in communication patterns can be performed in a controlled environment. Even though this method has a very high accuracy in detection of known botnets, it lacks the ability to detect unknown attacks, malicious encrypted traffic and it cannot easily find malware bots that use social engineering as propagation vectors [29].
2. **Signature-based detection**: This technique uses predefined patterns and communication signatures of known botnets, and common methods include regular

expressions, blacklists, and N-gram models. Feature matching rules are used in systems like Snort [29], which has a feature rich database, it can immediately detect the presence of malware bots and it doesn't create false positives. However this solution does not prevent zero-day botnet attacks from happening, even if they only have slightly different signatures [30] and code obfuscation technology makes the detection impossible [29].

3. **Anomaly-based detection**: Anomalies in the network traffic are used to detect bots, where unusual patterns may indicate its presence, like high traffic volume, increased latency and unusual system behavior [31]. This technique solves the problem of detecting unknown botnets and can be divided into *Host-based detection* and *Network-based detection*.

2.3.1 Host-based detection

This strategy monitors the internal behavior of computer systems, for instance, in terminal devices or/and in DNS Servers, by analysing their process activities, such as system and socket calls, registry changes, file system changes, etc, rather than on their external interfaces. Two examples of these techniques are the use of mining algorithms in multiple log files to identify C&C channels, where their communications are recorded in the host log file, used to detect normal and anomalous traffic [32], and the BotSwat, which relies in the similarities between the system call arguments and network data in order to predict if a given machine is being remotely controlled or not [30] [31].

Host-based techniques can cause however, performance issues, as they are constrained in terms of the host's computing and power resources, limiting the complexity and efficiency of anomaly detection algorithms that may harm their functionality. In small, low-powered end devices, like wearables, this technique is simply not feasible [33]. If one is using cloud services that are managed and maintained by third party providers, one is most likely to have restricted control over them, not being able to access key administrative tasks and services, such as updating the firmware and accessing the server shell, of which may incur multiple security problems and threats, for instance, data leakages, sharing of resources and inside attacks [34].

2.3.2 Network-based detection

The other botnet detection approaches may struggle to detect unknown botnets, to differentiate between the similarities of encrypted normal and anomalous traffic, or are not simply suitable for the heterogeneity of computer systems. Network-based detection, on the other hand, is a technique that tries to detect the presence of malware bots by solely monitoring network traffic behaviors, extracting their statistical and temporal characteristics, and hence being more robust to obfuscation and information concealing techniques. [31].

Botmasters normally create little traffic, other way their detection can led to persecution and detention. For that reason, botnet traffic is generated, most of the time, at the bot end, when bots are executing their assigned tasks, therefore, observing and monitoring traffic patterns at the end devices generally has greater success of detection. Another key aspect of botnet traffic is the degree of coordination that a certain malware bot has and the number of infected devices within the enterprise network. Malware bots with a greater number of automated tasks are harder to detect via network traffic analysis as they need to retrieve fewer commands from the C&C server in contrast to less autonomous bot devices, generating lower throughput [14]. If the number of bots in a network is low, their overall generated traffic is also low, being harder to detect.

Network-based detection can be divided into two groups: **active** and **passive monitoring**. Active monitoring is a technique that is based on injection of new packets into the network and measuring its reaction. It doesn't monitor actual users, but rather emulates real user behaviors with the drawback that it produces extra traffic. With active monitors one can see potential vulnerabilities before the malware infection, however, these network probes can cause interference with the normal traffic, having low preference among researches and network security professionals. Passive monitoring rely instead in packet traces, collected by network analyzers that capture live network traffic features, such as packet rates and packet timing, while generating less strain in networking systems [35][36]. These features can be posteriorly used to feed into algorithms that find unsusual behaviors and patterns in data. This class of algorithms have been widely studied in Statistics and Machine Learning, and are routinely named as anomaly detection algorithms.

Although network anomaly detection seems promising, one must be aware that real-world data might contain noise, which may indicate the presence of actual anomalies, and normal behaviors are continually evolving, therefore a continuous development of modern and more robust techniques needs to be taken in place [37].

2.4 Machine Learning

The searching for patterns in data has a long history of success [38]. In the 16th century, a wealthy scientist, known for dedicating his life to astronomical observations, Tycho Brahe, died, passing his lifetime observations to Kepler's hands, at that time an appointed astronomer. Kepler believed that the motion of the planets can be explained through mathematical theory and analytical reasoning, and Tycho's observations helped Kepler to found the empirical laws of planetary motion, leading to the development of classical mechanics [39]. In a similar way of thinking, Machine Learning algorithms try to infer mathematical and statistical models from regularities in observational data with multiple dimensions (features) extracting their analytical properties in a high-dimensional space, and taking actions by dividing that space into regions of categories, or classes.

In 1998, Tom Mitchell, a pioneer in the Machine Learning field, gave a modern definition of what is exactly Machine Learning: *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."*. In other words, a Machine Learning algorithm must generalize the experience in a way that allows to improve the machine performance on new, unseen data, without being literally programmed, other way, it is just a *hardcoded* solution.

2.4.1 Machine Learning workflow

When doing a Machine Learning project, there are a number of practices that are important to follow if one wants to achieve practical results. Starting from the **data collection** part, a dataset can be created with data gathered from multiple sources, like sensors, files, databases, etc, therefore ensuring the reliability of the sources is an important step for a good outcome of the model. Nevertheless, real-world data is, most of the time, messy, noisy, or simply missing, and hence **data pre-processing** needs to be done before it gets processed by Machine Learning models.

Different and heterogeneous sources, sometimes, can take data in various formats. For instance, data can be in a qualitative, or categorical, format, meaning that it is not expressed in the form of numerical values, but, instead, in the form of nominal or ordinal variables. They generally take this format to name, label, or categorize particular features. They must, however, be transformed into discrete quantitative values as Machine Learning models only accept qualitative data.

Missing data occur when certain values are missing in an observation, usually due to human error when processing data, machine error when there is some form of malfunction in the equipment, or when merging unrelated data. There are certain ways to deal with missing values. One can simply ignore them by removing the row, or column, where the missing value is located, however, this technique is only efficient when there aren't a lot of missing data points. Another technique of handling missing data is filling the empty entries with estimated values, such as the mean or the highest frequency value [40].

Data mining algorithms tend to assume that any dataset is free from disturbances [41]. These disturbances are expressed, most of the time, in the form of outliers, and can be treated with smoothing methods, such as the binning method, where the data is sorted by bins and replaced with the mean value, or the min-max value, of the bin; the regression method, where data can be smoothed by fitting it into a regression function;

and the clustering method, where outliers can be detected if they do not fit in any particular cluster [42].

The next step is to **research the model** that will fit best for the type of data. If the data is labeled, **supervised learning** algorithms will fit best for that particular problem as they work with labeled datasets, otherwise, **unsupervised learning** algorithms are more appropriate. If the problem requires integration with the environment and getting feedback from it, then one should consider **reinforcement learning** algorithms.

In classic supervised and unsupervised learning, models are typically trained with training sets and performance is validated through validation sets until satisfactory performance is achieved. After the model training, one more validation is needed for an unbiased performance estimation of the chosen Machine Learning algorithms. This extra validation step is made with test sets that contain unseen data and it helps getting a better sense on how the models will perform in unseen, real-world data.

2.4.2 Feature Selection

Feature Selection is a technique that aims to reduce the number of features by removing the redundant and irrelevant ones. Selecting only a subset of relevant features benefits in the reduction of computational costs and can improve the performance of the model. Feature Selection methods primarily perform the reduction of features with the help of the class value, being used to remove irrelevant features, although more robust techniques can also remove the redundancy between features, for instance, by using correlation [43].

Feature Selection methods can be divided into Filter, Wrapper and Embedded approaches:

2.4.2.1 Filter Approach

Filter approaches are called this way because they filter out less relevant features before a classification task. They are simple to implement and work by ranking the uniqueness of the input variables through some scoring criteria. A feature is said to be unique when it provides helpful information about a given class in the dataset. The variables that don't meet a certain threshold are regarded as irrelevant, being conditionally independent from the class labels, and, therefore, removed from the feature set.

Pearson correlation coefficient.

Pearson correlation coefficient is a Filter technique that measures the linear correlation between a feature and a class value, and is defined as:

$$R(i) = \frac{cov(X_i, Y)}{\sigma(X_i) \cdot \sigma(Y)} \tag{2.1}$$

Where x_i is the ith feature, Y the class label, *cov* and σ are the covariance between the two variables and their standard deviations. The result is a normalized value between -1 and 1.

Mutual Information.

Mutual Information (MI) measures the mutual dependence between two variables. It quantifies the amount of information that a variable obtained by observing another variable [44]. To understand MI, the concept of entropy needs to be known, and it is defined as:

$$H(Y) = -\sum_{y \in Y} p(y) \cdot log(p(y)) \tag{2.2}$$

The equation 2.2 represents the uncertainty of the variable Y. If the entropy, or uncertainty, of Y is measured under the observation of a given variable X, then it is defined as

$$H(Y|X) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \cdot log(p(y|x)) \tag{2.3}$$

Where $p(x,y)$ is the joint probability of x and y, and $p(y|x)$ is the conditional probability distribution. The decrease in uncertainty is given by:

$$I(Y,X) = H(Y) - H(Y|X) \tag{2.4}$$

And it is defined as the MI between Y and X. Another way of writing MI is to define it using the *Kullback-Leibler Divergence* between the joint probability and the product of the marginal distributions $p(x)$ and $p(y)$:

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \cdot log \frac{p(x,y)}{p(x)p(y)} \tag{2.5}$$

If the MI between Y and X is zero, they are independent, otherwise they have some degree of dependency, meaning that one variable can provide information about the other.

Other statistical tests can be used in feature ranking. For instance, Chi-Square measures divergence from the expected distribution if the feature under observation is assumed to be independent from the class value [45].

The downsides of using some Filter methods like the Pearson correlation coefficient and MI are that they do not correlate between features, of which can be highly correlated with each other, and features that are less relevant to the class value by themselves, hence being discarded, can be relevant when combined with other features.

2.4.2.2 Wrapper Approach

Wrapper methods use the predictor model as a black box, performing a greedy search by evaluating all the possible subsets and using the model performance as their objective function. Since *brute-forcing* all the combinations is a NP-hard problem, the number of subsets grows combinatorially. As a result, heuristic algorithms are used to speed-up the selection process and some of them are going to be presented below.

Sequential forward selection.

The Sequential forward selection (SFS) algorithm starts with an empty set of sub-optimal features. In the first step, single features are tested and the single best feature is selected using the objective function. The single best feature is used to form a pair with a feature picked from the remaining features and the best pair is selected. This procedure continues until the required number of features are selected.

Sequential backward selection.

The Sequential backward selection (SBS) algorithm is similar to SFS algorithm, although it starts from a subset containing all the features. In the first step, all the subsets containing $n-1$ features are tested and the worst performing feature is removed. This procedure continues until the required number of features are left.

Genetic Algorithms.

Genetic Algorithms (GAs) are stochastic methods that aim to mimic the behavior of genetic evolution, and can be used to find the best suboptimal subset of features. Features are encoded as chromosome bits (1 being active and 0 being inactive), the model performance is the fitness function and the next generation can be created by crossing-over and mutating the features of the best subsets.

Particle Swarm Optimization.

Like GAs, Particle Swarm Optimization (PSO) is a nature-inspired optimization algorithm that came from the observations in social behaviors of natural systems. The candidate subsets are called particles and the algorithm works on the concept that the position of a particle is adjusted by a mathematical formula that takes into account its local best known position and the swarm's best known position. In this case, the positions are each subset of features and the best known position can be calculated with the model performance.

Recursive Feature Elimination.

Recursive Feature Elimination (RFE) is a Feature Selection method that ranks features by their coefficients, or importance, and simply removes the least important feature, one by one, until the optimal number of features is left. Features are ranked using some estimator, such as Support Vector Machines (SVM) or Decision Trees[46], and by recursively eliminating a small number of features per iteration, RFE attempts to eliminate irrelevant features and collinearity that may degrade the model's performance.

Using Wrapper approaches have some drawbacks [43]. If the number of samples is large, the majority of time is spent in training the model, rather than choosing the best subset of features, thus being computationally expensive. In GAs, the same subset of features might be evaluated multiple times since they are memoryless. Overfitting can also occur if the predictor model has poor generalization during the test set. Limitations in SFS and SBS can be found when highly correlated features are included in the suboptimal subset if they gave the highest performance in the objective function. Alternative implementations of SFS and SBS propose to solve this problem by adding additional phases to reduce the maximum number of features, and statistical distance tests as objective functions to obtain less redundant features. For instance, MI can be used as an objective function for greedy search algorithms and can be designed in a way that the MI between the selected feature and the subset is minimized. This is formulated as:

$$I(Y, f) - \beta \sum_{s \in S} I(f; s) \tag{2.6}$$

Where Y is the class label, f is the current selected feature, s is a feature in the selected subset S, and β a parameter to control the weight of the MI between the selected feature and the subset.

2.4.2.3 Embedded Approach

Embedded methods intent to perform feature selection during the model's training phase, being embedded in the algorithms themselves. This approach is faster than Wrapper methods because the feature selection is done in parallel, rather than retraining the algorithm with different subsets [47].

Regularization models.

Embedded mechanisms can be expressed in the regularization form and are commonly used in Least square solutions for linear problems. They induce penalties to features that don't contribute to the model, by reducing its coefficients to small values, or to zero values. Examples of regularization models are:

1. **Ridge regression**: penalizes the sum of squares, where larger weights contribute more to the penalization, hence penalizing features that have larger weights;
2. **LASSO regression**: penalizes the sum of absolutes, and so, small weighted coefficients may reach the zero value, vanishing their feature. This technique has

some limitations. If there is a group of highly correlated features, LASSO tends to select only one feature, ignoring the others [48].

3. **Elastic Net**: is a weighted combination of Ridge and LASSO regressions and pretends to overcome the limitations of LASSO regression by not easily eliminating highly correlated features.

A graphical representation of the afore-mentioned regularization methods can be seen in Figure 2.6.

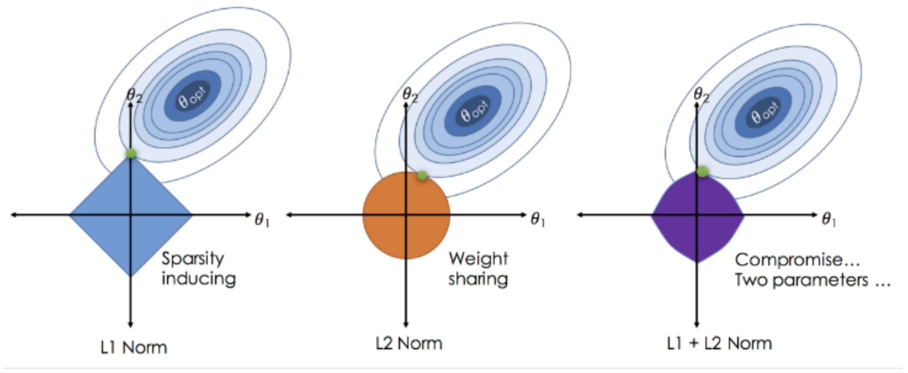


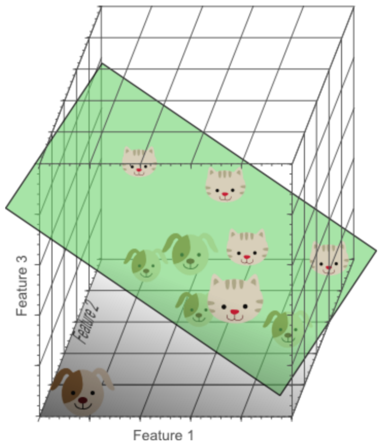
Figure 2.6: Lasso, the Ridge and the Elastic Net Regressors in comparison. Source: Regularization and variable selection via the elastic net [49]

Feature Importance.

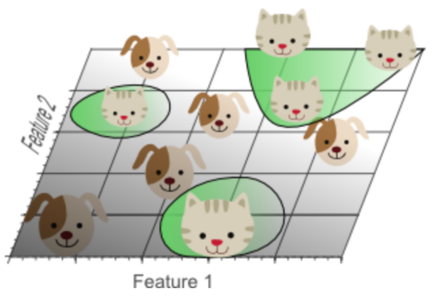
Feature Importance is generally related to Tree-based classifiers as they split a dataset based on its features, i.e., ranking them by how well they improve the purity of the node. This particularity makes Tree-based algorithms suitable, not only for classification and regression tasks, but also for feature selection tasks. Random Forests are a prime example of an effective feature selector based on Feature Importance, where decision trees vote for the one that contains the best subset of features.

2.4.3 Dimensionality Reduction

Dimensionality Reduction is a process of transforming high dimensional data in low dimensional representations, aiming to address the *Curse of Dimensionality* problem while maintaining as much of the variance as possible [50]. Compressing the number of dimensions in a dataset has several benefits, as it can lead to lesser training times and memory usage, faster model convergence, better data visualization, and even greater model performance. This occurs because increasing the number features, or dimensions, while maintaining the size of the training set, decreases exponentially the density of training samples. Intuitively, having sparser data increases the odds of finding a separable hyperplane, as for a given training sample, the likelihood of being in the wrong side of the best hyperplane is infinitely small when the number of dimensions are infinitely large. However, the model starts to learn exceptions to that training set so that an infinite amount of new and unseen data may not fit into these exceptions, i.e. we say that the model is **overfitting** the training data [51].



(a) More features increases the changes of having a perfect separable hyperplane



(b) If we project the perfect separable hyperplane onto a 2D space, we can see that the classifier starts to overfitting.

Figure 2.7: An example of the Curse of Dimensionality. Source: The Curse of Dimensionality in classification [51]

2.4.3.1 Principal Component Analysis

Principal Component Analysis (PCA) is the most widely used dimensionality reduction technique and it works by finding the basis vectors of the orthonormal sub-space where data vary the most [52]. In Statistics, the variance is a measure of dispersion, revealing how data is spread out from their average value. The dispersion of each pair of variables in the feature space can be calculated by the covariance matrix [53]:

$$S = \frac{1}{n-1} BB^T \tag{2.7}$$

Where each column of a matrix B, with dimensions $m \times n$, contains the deviations of each sample from its mean, being represented as follows:

$$B = |\overrightarrow{x_1} - \overrightarrow{\mu}|...|\overrightarrow{x_n} - \overrightarrow{\mu}| \tag{2.8}$$

With $\overrightarrow{x_i}$ being the sample vector and $\overrightarrow{u}$ being the mean of the column.

A matrix is said to be positive semi-definitive if it can be obtained as the product of a matrix by its transpose and it is always symmetric. Looking at equation 2.7, one can see that BB^T respects this assumption, and so, the covariance matrix is symmetric and positive semi-definitive, meaning that it can be orthogonally diagonalized, its eigenvalues are always real, non-negative numbers and its eigenvectors are pairwise orthogonal when their eigenvalues are different.

The eigen-decomposition is important to PCA because the principal components (PCs) are eigenvectors that can be extracted from the linear combinations of the features in a dataset, selecting those with the largest eigenvalues that point in the direction to the largest variance, while being orthonormal (PCs that have different eigenvalues), or else the transformation can be heavily disturbed. The Figure 2.8 exemplifies a PCA transformation to a lower coordinate system where each axis is a principal component.

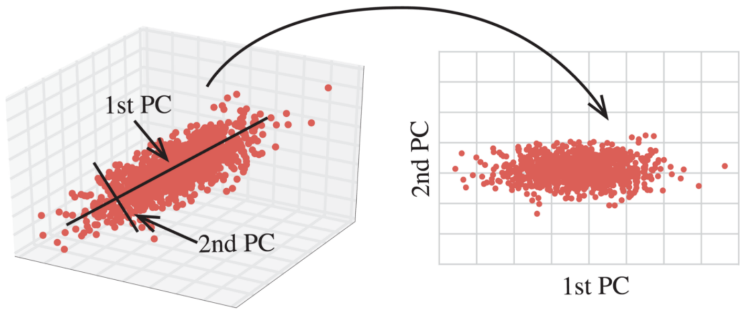


Figure 2.8: 3D to 2D space transformation using PCA

2.4.4 Anomaly Detection

Anomaly Detection (AD) is the process of identifying patterns in data that differ greatly from the expected behavior, and it is routinely used in fraud detection, cyber-security, and healthcare, among other areas. These unusual observations that deviate considerably from the normal distribution are often called as anomalies, and they might indicate a potential threat to the resilience and security of any system that is being monitored [54]. Their detection, although, can be challenging. For instance, the variability of legitimate behaviors can be very large, and anomalies themselves can be very diverse, in a way that makes a predictor to not identify correctly the anomalous data, or to have a high false alarm rate, thus proper preprocessing, normalization, and feature selection or dimensionality reduction methods are important steps to take to improve detectability.

In a statistical context, an anomaly can be defined as a data point x, or a group of points, contained in a data space X that lies in a low probability region under a distribution $\mathbb{P}^+$, representing a normal behavior and described with some **pmf!** $p^+(x)$. Consequently, a set of anomalies can be formalized as

$$A = \{x \in X | p^+(x) \leq \tau\}, \tau \geq 0 \tag{2.9}$$

where the probability of A under $\mathbb{P}^+$ is lower or equal than some defined threshold τ [55].

Anomalies can also be categorized in three major groups, based on their nature [54]–[56]:

1. **Point anomaly**: a individual anomalous data point that differs from the standard behavior. For instance, an illegal transaction in a fraud detecting system, or a unique cat image in a dataset of dog images.
2. **Contextual anomaly**: a data point that is considered an anomaly in some particular context. As an example, a value of 100 Km/h is considered an abnormal velocity when tracking the speed of a bike, but typical when the tracking object is a car. In this type of anomalies, the distribution representing the normal behavior is better defined as a conditional distribution $\mathbb{P}^+_{x|T}$, described with some pdf, $p^+(x|t)$, that depends on some contextual variable T.
3. **Collective anomaly**: a subset of data instances that might be normal by themselves, but their occurrence as a collection is considered anomalous with respect to the entire dataset, when related together, such as similar, or related, botnet attacks.

With the advent of Deep Learning methods for AD, *Ruff et al.* [55] suggested the creation of two new groups of anomalies targeted by Deep AD models:

1. **Sensory (low-level) anomalies**: deviations that occur in the low-level feature hierarchy, such as edges in images and individual words in semantic concepts and topics in text. Examples of low-level anomalies could be character typos in text, and texture defects in images of clothes.
2. **Semantic (high-level) anomalies**: samples that don't fit in the normal class are inferred as samples that came from a different, anomalous class. Some anomalies can be indistinguishable from normal instances in the raw or low-level feature spaces, but when transformed to higher-level feature hierarchies, divergent patterns start to emerge. For instance, a dog with a a texture and color similar to a cat can be confused with a cat in the raw feature space, but be easily recognizable as a dog to the discriminative classifier when transformed to a higher level feature space.

The Figure 2.9 illustrates the different types of anomalies aforedescribed that can be encountered when solving different AD tasks.

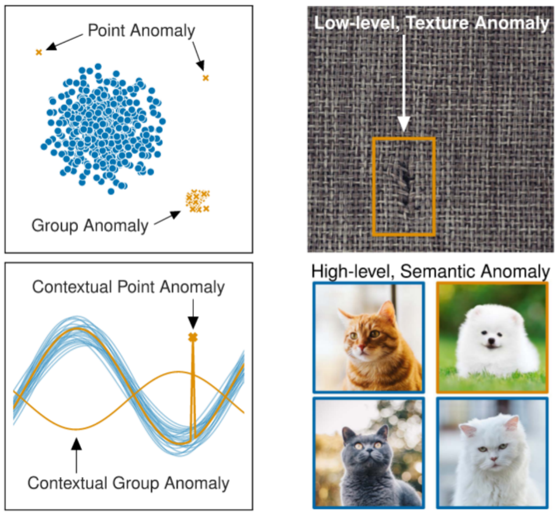


Figure 2.9: Different types of anomalies. Source: A Unifying Review of Deep and Shallow Anomaly Detection [55]

2.4.4.1 AD approaches

The availability of class labels impacts in the choice of Machine Learning models for AD. Therefore, AD methods can be divided into three categories [55], [56]:

1. **Supervised AD**: this approach assumes that the entire dataset is labeled, hence it tries to build a predictive model that discriminates between normal and anomaly classes, and requires a balanced dataset and a accurate representation of anomalous classes. Although supervised settings are faster to compute and usually achieve higher accuracy in the given classes, they normally are not used in a real world scenario where one does not have access to all types of anomalies.
2. **Unsupervised AD**: models from this category are, arguably, the most commonly used in AD tasks, due to the fact that these algorithms do not require training data, and take into account that anomalies are rare to find and costly to generate, providing lower false alarms when these assumptions are true. They suppose that one cannot ensure accurately the *ground-truth* level sets of $\mathbb{P}^+$, and so, they assume that data is a mixture of normal and abnormal instances with an estimated contamination rate $\eta \in (0, 0.5]$. A downside of this estimation is that the contamination present in the dataset makes the decision boundary more prone to be distorted.
3. **Semi-supervised AD**: Semi-supervised AD algorithms attempt to build a model around the normal behavior, and use that model to identify abnormal instances in the test data. They can use unlabeled anomalous samples in the training phase together with the normal class labeled, as well as training with only the normal instances, in an unsupervised manner. The latter is usually preferred as it is less

prone to overfitting due to the scarcity of anomalous samples and the outputed models are usually more robust in detecting anomalies, even if they are only a few, nonetheless data must be noise-free and have no contamination whatsoever.

2.4.4.2 Anomalies, Outliers and Novelties

Although the vast majority of the literature treat anomalies, outliers and novelties as being the same, some studies [55], [56] argue that these terms have subtle differences: an anomaly can be characterized as being a data entity from a distribution other than $\mathbb{P}^+$, for instance, if $\mathbb{P}^+$ represents a distribution of dogs, then a cat is from a different distribution $\mathbb{P}^-$, i.e. an anomaly. An outlier, on the other way, can be considered a rare instance from $\mathbb{P}^+$ that happens to lie in a low probability region, and a novelty, an instance from a new region of a non-stationary distribution $\mathbb{P}^+$. For instance, a rare breed of dogs is considered as an outlier, and a new of dogs a novelty.

Despite the differences in terminology, methods for detecting anomaly, outlier and novelty instances are mostly the same, nevertheless, outlier detection estimators usually are unsupervised, require a polluted dataset, i.e. a dataset with outliers, and assume that outliers/anomalies cannot create a dense cluster but be in low density regions, whereas novelty detection algorithms must be trained with a pollution-free set of normal instances and novelties/anomalies can create a dense cluster as long as they are in a low probability region under the trained distribution $\mathbb{P}^+$ [57].

Because these terms are used interchangeably, and because we have a perfect *ground-truth* about what is normal and what is not, as one can see throughout the Chapter 3, in this thesis only Novelty Detection algorithms will be considered, and for simplification purposes, they are going to be referred to as AD algorithms.

2.4.4.3 Density-based models

The most straightforward approach to perform AD is to compute the Mahalanobis distance, from a test point to the distribution $\mathbb{P}^+$, i.e. how many standard deviations that point is from the mean, classifying it as a normal instance or an anomaly based on a defined threshold value [55]. This is analogous to assume that the training data is generated by some underlying probability distribution, of which can be estimated during the training phase, evaluating the log-likelihood of a test sample in conformity with that model.

Density estimation techniques can be divided into two major groups: the *parametric* methods, where they fit a given model on data assuming that data come from a population that can be modelled by a probability distribution with a finite set of parameters, and *non-parametric* methods, where they do not make assumptions about the underlying probability distribution, and so, they can adapt very easily to the complexity of data that do not conform to a particular known distribution [58].

Multivariate Gaussian model.

The simplest model that one can assume that data is generated from is the Gaussian distribution (Figure 2.10). Normal distributions are observed commonly in nature. Even though populations might not follow a Gaussian distribution frequently, the distribution of its sample means, most often, will [59]. This probability distribution for a d-dimensional object $\boldsymbol{x}$ of some data X is denoted as follows:

$$p_{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right\} \tag{2.10}$$

where $\boldsymbol{\mu}$ is the mean, and Σ is the covariance matrix. With this approach, the most

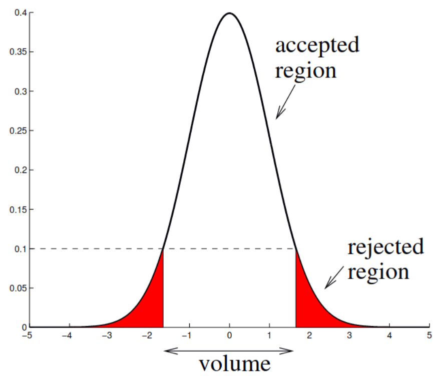


Figure 2.10: 1-dimensional Gaussian distribution with a defined threshold. Source: One-class classification [60]

costly computational operation is the inversion of the covariance matrix [60], and to estimate the optimal parameters of the assumed Gaussian distribution, one can use the Maximum Likelihood Estimation (MLE) that is defined as:

$$\begin{aligned} l(\boldsymbol{\mu}, \Sigma; X) &= \arg\max_{\boldsymbol{\mu},\Sigma} \prod_{i=1}^{m} p_{\mathcal{N}}(\boldsymbol{x}^{(i)}; \boldsymbol{\mu}, \Sigma) \\ &\Rightarrow \arg\min_{\boldsymbol{\mu},\Sigma} -\sum_{i=1}^{m} \log p_{\mathcal{N}}(\boldsymbol{x}^{(i)}; \boldsymbol{\mu}, \Sigma) \end{aligned} \tag{2.11}$$

The advantages of using logarithms to compute the MLE are that they require less computational resources since one can rewrite the product of the likelihoods as the sum of the log likelihoods, of which is faster to compute, and taking the product of likelihoods would become very small, that one would run out of floating point precision, yielding an underflow. One can also convert the MLE into a minimization problem by taking the sum of the negative log likelihoods (Equation 2.11) and finding a closed-form solution for the estimates of the multivariate Gaussian parameters $\boldsymbol{\mu}$ and Σ:

$$\begin{gathered}\frac{\partial l(\boldsymbol{\mu}, \Sigma; X)}{\partial \boldsymbol{\mu}} = 0 \\ \Rightarrow \hat{\boldsymbol{\mu}} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}\end{gathered} \tag{2.12a}$$

$$\begin{gathered}\frac{\partial l(\boldsymbol{\mu}, \Sigma; X)}{\partial \Sigma^{-1}} = 0 \\ \Rightarrow \hat{\Sigma} = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \hat{\boldsymbol{\mu}})(\boldsymbol{x}^{(i)} - \hat{\boldsymbol{\mu}})^{T}\end{gathered} \tag{2.12b}$$

Gaussian Mixture Models.

The underlying distribution of most datasets are not convex and unimodal. For those cases, one way of getting a more flexible model is to mixture a finite number of normal distributions, forming a linear superposition of Gaussians [60]:

$$p(x) = \frac{1}{N} \sum_{j} \alpha_j p_{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu_j}, \Sigma_j) \tag{2.13}$$

where N is the number of Gaussian components, and α_j are the mixing coefficients. When number of Gaussian components is defined beforehand, the means and covariances can efficiently be estimated using the Expectation-Minimization (EM) algorithm, which is a powerful method of finding maximum likelihood solutions for models with latent variables [38].

When the training set is sufficiently large, this method works very well, showing a smaller bias in comparison to the single Gaussian distribution, however it requires a large number of training samples to overcome the Curse of Dimensionality, and thus shows more variance with a limited amount of samples. An illustration of a Mixture of Gaussians is depicted in Figure 2.11.

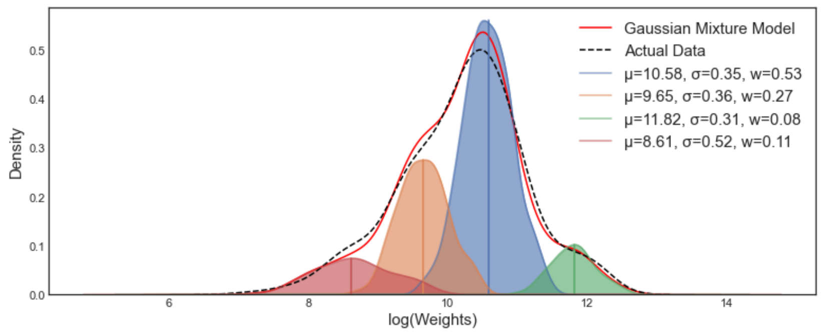


Figure 2.11: Mixture of Gaussians

Kernel Density Estimation.

Kernel Density Estimation (KDE), also known as Parzen Windows Estimation, is a non-parametric way of estimating the pdf of random variables. Being non-parametric, KDE algorithms do not make assumptions about the underlying distribution of the training data, allowing they to naturally follow the shape of data, like a histogram would, but using kernels to smoothen the distribution. The generalized formula for KDE is as follows:

$$p(x) = \frac{1}{nh}\sum_{i=1}^{n}\phi(\frac{x - x_i}{h}) \tag{2.14}$$

where ϕ is the Kernel method, and h is a smoothing parameter, called bandwidth. The Kernel method is placed on each data point and then a sum of the local contributions of each kernel is calculated. Often, ϕ is represented as a standard gaussian with mean zero and variance 1, centered on each training point [58], [61], and so, the only parameter that one needs to estimate is the bandwidth, making the training costs are almost zero. On the contrary, the testing is expensive, as all training examples have to be stored to posteriorly compute and sort the distances from all testing objects to all training objects.

Given that non-parametric methods try to build a model around the training data, they also suffer from the *Course of Dimensionality*, where the number of data samples to provide the same spatial resolution required to model an estimator in a lower dimension grows exponentially, however they are a powerful method of modelling a representation of normality.

2.4.4.4 Boundary-based models

Estimating a complete data density in an effort to construct a representation of normality might be too demanding, specially when the data to do so is scarce and only the data boundaries are enough. Therefore, Boundary-based methods for AD optimize a closed boundary around the training set. The volume of the boundary is not always minimized, but most algorithms have a strong bias towards minimal volumes. Although Boundary-based methods require less samples than Density-based methods, they rely heavily on the distances between objects, therefore, data scaling plays an important role in the viability of these methods.

Nearest Neighbor.

The Nearest Neighbor method can be derived from a local density estimation by the k-Nearest Neighbours (kNN) algorithms. It avoids explicit density estimation, using only the Euclidean or Mahalanobis distances to the first nearest neighbor. One advantage of using this algorithm is that, like the KDE, it does not require to construct a model of normality since the training data points are stored to later measure the distance to a given a test point. However, it can be computationally expensive during the test phase. In the nearest neighbor density estimation a cell, often an hypersphere in d dimensions, is centered around the test point $\boldsymbol{z}$. The volume of the cell grows until it *engulfs* k data points from the training set. The local density can be estimated using the following formula:

$$p_{NN}(\boldsymbol{z}) = \frac{k/N}{V_k(||\boldsymbol{z} - NN_k^{tr}(\boldsymbol{z})||)} \tag{2.15}$$

where $NN_k^{tr}(\boldsymbol{z})$ is the k nearest neighbor of $\boldsymbol{z}$ in the training set and V_k the volume of the cell engulfing that training point.

In the One-Class classifier, a new test point belongs to the normal class if its larger or equal to the local density of its nearest neighbor in the training set, with $k = 1$:

$$f_{NN^{tr}}(\boldsymbol{z}) = p_{NN}(\boldsymbol{z}) \geq p_{NN}(NN^{tr}(\boldsymbol{z})) \tag{2.16}$$

This is equivalent to:

$$\begin{aligned} f_{NN^{tr}}(\boldsymbol{z}) = & \frac{V(||\boldsymbol{z} - NN^{tr}(\boldsymbol{z})||)}{V(||NN^{tr}(\boldsymbol{z}) - NN^{tr}(NN^{tr}(\boldsymbol{z}))||)} \leq 1 \\ \Rightarrow & \frac{||\boldsymbol{z} - NN^{tr}(\boldsymbol{z})||}{||NN^{tr}(\boldsymbol{z}) - NN^{tr}(NN^{tr}(\boldsymbol{z}))||} \leq 1 \end{aligned} \tag{2.17}$$

Meaning that the distance from the test point $\boldsymbol{z}$ to its nearest neighbor in the training set $NN^{tr}(\boldsymbol{z})$ is compared with the distance from this neighbor $NN^{tr}(\boldsymbol{z})$ to its nearest neighbor.

Local Outlier Factor.

The Local Outlier Factor (LOF) is an algorithm that estimates the local density of test points to data points that are its neighbors. By comparing these, we can determine which data points have similar densities, and which have substantially lesser density than its neighbors, being considered as outliers. However, this algorithm differs from the kNN algorithm because kNN do not work well when the training set has different density regions, and so, LOF is more robust to outliers [62].

If we consider the *k-distance* of an object o, denoted as $dist_k(o)$, as the distance to its *k-th* nearest neighbor, and its *k-distance neighborhood* denoted as $N_k(o) = o'|o' \in D, dist(o, o') \leq dist_k(o)$, then the local reachability density lrd of o is defined as:

$$lrd(o) = \frac{||N_k(o)||}{\sum_{o' \in N_k(o)} reachdist_k(o' \leftarrow o)} \tag{2.18}$$

where, $reachdist_k(o' \leftarrow o) = maxdist_k(o), dist(o, o')$ is the reachability distance, i.e., the maximum distance of two points and the *k-distance* of that point. The local reachability density is a way of computing the density of k-nearest points around a point by taking the inverse of the sum of all of the reachability distances of all the *k-nearest* neighbors.

Finally, the LOF can be calculated as the average ratio of local reachability of o and its k-nearest neighbors:

$$LOF_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{lrd_k(o')}{lrd_k(o)}}{||N_k(o)||} \tag{2.19}$$

This means that if the density of the neighbors is lower than the density of o, then the point o is an inliner, otherwise it would mean that the point o is an outlier.

One-Class Support Vector Machines.

One-Class SVM, or ν-SVM, is the unsupervised version of Multi-Class SVM that learns to minimize the volume of the data-enclosing hyperplane of the single class of instances in training data, considering all the other samples outside that hyperplane as outliers. For both One-Class and Multi-Class paradigms, SVMs are able to classify instances of classes by projecting the data points to a higher-dimensional space using a non-linear function ϕ. With this kernel function, data points can be linearly separated by an hyper-plane defined as $w^T + b = 0$ in a higher dimensional feature space F that determines the margins from the origin. The objective function of the One-Class SVM is defined by the following minimization problem [63]:

$$\begin{gathered} min_{w,b,\xi_i} \frac{||w||^2}{2} + \frac{1}{\nu n} \sum_{n}^{i=1} \xi_i - \rho \\ \text{subject to :} \\ y_i(w^T \phi(x_i) + b) \geq \rho - \xi_i \qquad \text{for all } i = 1, ..., n \\ \xi_i \geq 0 \qquad \text{for all } i = 1, ..., n \end{gathered} \tag{2.20}$$

where ν is a regularization term that sets a upper bound on the number of samples that lie on the wrong side of the hyperplane, and thus being considered as outliers , and a lower bound on the number of samples used as Support Vector.

2.5 Summary

In this chapter, it was understood how malware bots can establish a C&C channel with C&C servers by using some rogue DNS server, and how they can be part of a larger network of compromised devices, called botnet. Anomaly-based Detection techniques can be effective in detecting malware bots and botnets activities, as well as as unknown behaviors. However, by using them in terminal devices, they are constrained to the host's limited computing resources. Network-based Anomaly Detection approaches are more suitable to the heterogeneity of computer systems, and therefore they are the preferred method for this dissertation.

Anomaly Detection uses Machine Learning algorithms to detect anomalies in data, meaning that a proper workflow must be followed. In this workflow, data plays the most important role as the way it is processed greatly impacts the performance of these models. In Anomaly Detection, and specially in Novelty models, one must be careful in using Feature Selection techniques, as these techniques use the target value to remove irrelevant and redundant features. Moreover, Novelty Detection methods only train with normal instances. Feeding these algorithms with non-clean data will negatively impact their performance.

In the next chapter, robust and privacy-focused mechanisms will be proposed, with proper data collection and processing, using proper unsupervised dimensionality reduction techniques that do not using the class values and with Novelty Detection models that are able to detect even zero-day attacks of malware bots that use the DNS to perform their illicit activities.

CHAPTER 3

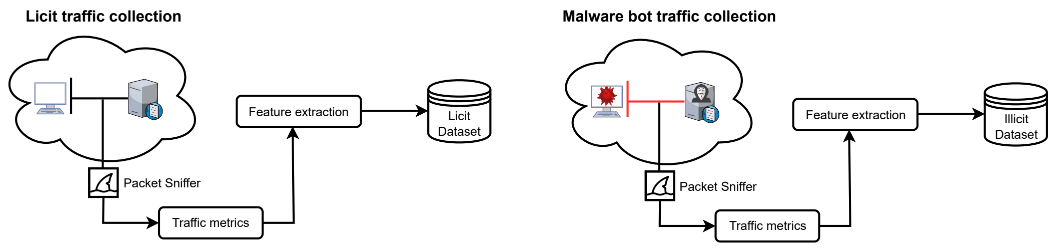
Methodologies for Network-based AD

As stated throughout the chapters 1 and 2, DNS is an important and vital communication channel for all Internet applications and organizations. The fact that it must be allowed in enterprise networks for a proper function of services and business processes, entails the necessity of actively monitoring its usage, otherwise it can be misused for illicit purposes, such as data exfiltration and C&C communications. The large majority of current solutions for AD perform DPI techniques, processing unnecessary personal information, that might cause privacy concerns from the data subjects, encasing the organization in privacy related issues and large fines. Moreover, in circumstances where the DNS communication is encrypted, these approaches would be useless.

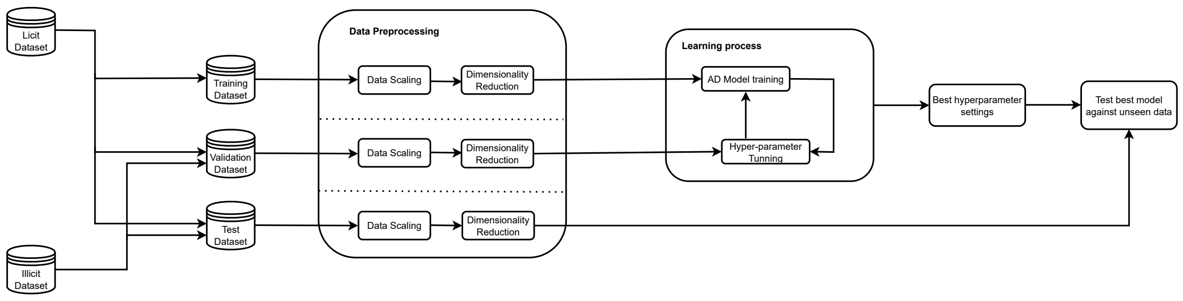
Therefore, this section proposes a privacy-focused framework for Network-based AD, where it processes as minimal personal and identifiable information as possible, while still successfully extracting relevant features for the modelling of semi-supervised AD algorithms.

3.1 Network-based AD workflow

Considering that Machine Learning algorithms are sensitive to the quality of data, all Machine Learning projects need to follow a number of steps, if the goal is to achieve satisfactory results. These number of steps, although, might be slightly different, according to the type of Machine Learning models that one is working with. In the case of semi-supervised learning algorithms for AD, the proper way to train them is to feed them with only normal instances. Once a concept of normality is learnt, that model is validated, and posteriorly tested, with datasets that contain a mixture of normal instances and anomalies. For bot malware detection using Network-based AD approaches, normal instances and anomalies are a product of preprocessed features, extracted from metrics collected from licit and bot malware traffic traces, respectively. A visual representation of a proper Network-based AD workflow for semi-supervised algorithms can be seen in Figure 3.1.



(a) Data collection and feature extraction



(b) AD pipeline

Figure 3.1: Network-based AD workflow using semi-supervised algorithms

3.1.1 Data collection

As stated before, semi-supervised AD techniques often try to reconstruct a model that represents a normal behavior, classifying a test point that deviates from this representation of normality as a novel instance, or an anomaly. In this case, the normal behavior is all the possible forms that a licit usage of the DNS can take, the novel instances must be observations extracted from malware activity, and the goal is to find out if the DNS is being used for malicious purposes, flagging all the DNS traffic generated by a given machine that show an abnormal pattern as anomalies, inferring afterwards that it is infected.

Using open datasets with licit and malware communication traces can be challenging. Public DNS datasets with C&C and data exfiltration traces can be limited in variety, the reliability of the *ground-truth* is hard to be ensured, since licit and illicit DNS traffic can be encrypted, and new normal and malware bot activity patterns are continually emerging as Internet technology evolve. Labeling samples with mixed traffic is also a complex and time-consuming task, since one often needs to combine DPI techniques and expert knowledge for a successful labeling, and yet, they cannot ensure that network attacks are fully identified [64].

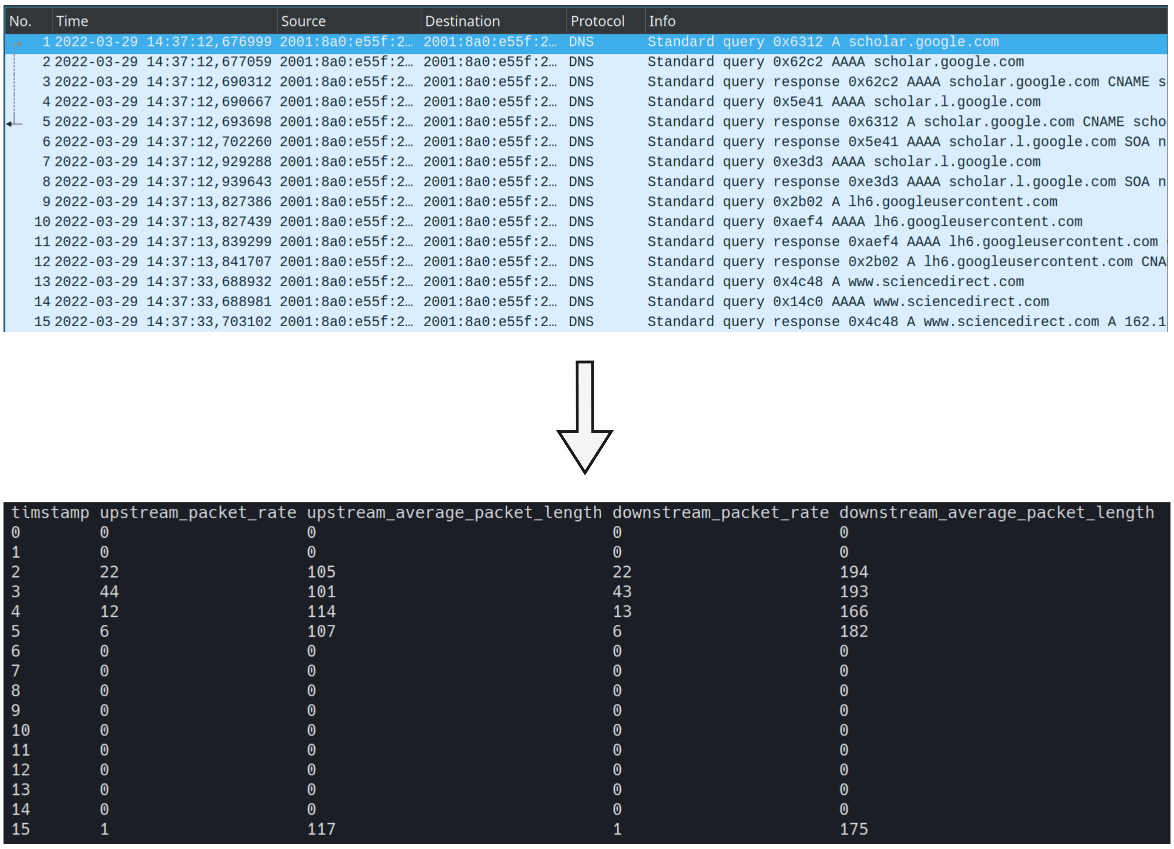
A better approach for a perfect *ground-truth* is to emulate malware traffic in a Laboratory environment, providing reliable data to Machine Learning models, where their predictive capacity could, otherwise, be degraded upon their deployment in a real world scenario, if malware bot attacks and normal behaviors were mislabeled. This emulation must gather a vast variety of possible illicit communication behaviors over multiple malware bot activities. Furthermore, the licit traffic must also cover a wide range of possible normal behaviors, and it must be produced under a controlled environment to ensure that the terminal device is not infected with some sort of virus, worm, Trojan horse or an active botnet.

As Figure 3.1a illustrates, the capture of both licit and illicit traffic can be done with software or hardware devices, called packet sniffers, that can intercept and log traffic data over a link in the network. This raw data is, most of the time, expressed in a qualitative format, and if not, some of them may be categorical, in the form of nominal variables. Machine Learning models don't know how to process such non-numerical data, meaning that the raw data must be transformed into quantitative forms. In network traces, this transformation can be done by measuring, or counting, the events of interest with a frequency of Δ, being Δ the sampling interval, or the timestamp resolution, resulting in a discrete time series of multiple metrics.

3.1.1.1 Data collection under data protection regulations

Botnet detection often deals with personal information, as it is needed to successfully detect and disrupt the C&C infrastructure employed by the botmaster. There are different techniques that researchers and cyber-security professionals can use to track botmaster activity, as well as infected devices, and they were discussed earlier in Section 2.3. In our case, we are only interested in identifying the misuse of the DNS to establish covert channels, or to exfiltrate data, by solely monitoring the DNS communications of individual terminal devices. This approach, despite the fact that it is robust to concealing techniques and more privacy friendly than the other solutions, still requires us to process IP addresses, of which are classified as Personal Identifiable Identification (PII), as they can reveal personal data such as geographical localization of individuals [65]. This collection is lawful in regulations like the GDPR if it is contemplated in a contract or under legitimate interest of improving the provided service. In contractual agreements, the data controller must inform the purpose of processing such collected data, from both internal and external sources, and if additional processing is made, that information has to be provided to the data subject as well. Once the personal information is stored, companies must ensure its integrity and confidentiality, and the data subject must have the right to request, obtain and destroy their personal information, including data provided to third parties for processing.

For the reasons aforedescribed, and because one should use as minimal PII as possible, the proposed collection of traffic metrics for malware bot detection uses IP addresses to solely differentiate if a given processed IP is from a set of known, and permitted, DNS servers or is an end device, which is turn, is useful for the collection of upstream and downstream network metrics, such as, the upstream and downstream metrics of packet rates and average packet lengths in each sampling interval. The Figure 3.2 depicts the proposed network metrics computed from a network trace and the Algorithm 1 presents an implementation for their computation.



```
No. Time                       Source             Destination        Protocol Info
  1 2022-03-29 14:37:12,676999 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0x6312 A scholar.google.com
  2 2022-03-29 14:37:12,677059 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0x62c2 AAAA scholar.google.com
  3 2022-03-29 14:37:12,690312 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query response 0x62c2 AAAA scholar.google.com CNAME s
  4 2022-03-29 14:37:12,690667 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0x5e41 AAAA scholar.l.google.com
  5 2022-03-29 14:37:12,693698 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query response 0x6312 A scholar.google.com CNAME scho
  6 2022-03-29 14:37:12,702260 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query response 0x5e41 AAAA scholar.l.google.com SOA n
  7 2022-03-29 14:37:12,929288 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0xe3d3 AAAA scholar.l.google.com
  8 2022-03-29 14:37:12,939643 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query response 0xe3d3 AAAA scholar.l.google.com SOA n
  9 2022-03-29 14:37:13,827386 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0x2b02 A lh6.googleusercontent.com
 10 2022-03-29 14:37:13,827439 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0xaef4 AAAA lh6.googleusercontent.com
 11 2022-03-29 14:37:13,839299 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query response 0xaef4 AAAA lh6.googleusercontent.com
 12 2022-03-29 14:37:13,841707 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query response 0x2b02 A lh6.googleusercontent.com CNA
 13 2022-03-29 14:37:33,688932 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0x4c48 A www.sciencedirect.com
 14 2022-03-29 14:37:33,688981 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query 0x14c0 AAAA www.sciencedirect.com
 15 2022-03-29 14:37:33,703102 2001:8a0:e55f:2… 2001:8a0:e55f:2… DNS      Standard query response 0x4c48 A www.sciencedirect.com A 162.1
```

```
timstamp upstream_packet_rate upstream_average_packet_length downstream_packet_rate downstream_average_packet_length
0        0                    0                              0                      0
1        0                    0                              0                      0
2        22                   105                            22                     194
3        44                   101                            43                     193
4        12                   114                            13                     166
5        6                    107                            6                      182
6        0                    0                              0                      0
7        0                    0                              0                      0
8        0                    0                              0                      0
9        0                    0                              0                      0
10       0                    0                              0                      0
11       0                    0                              0                      0
12       0                    0                              0                      0
13       0                    0                              0                      0
14       0                    0                              0                      0
15       1                    117                            1                      175
```

Figure 3.2: Raw network traces to network metrics

Algorithm 1 Collection of network metrics from network traces

```
Require: $capture$, $nameservers$
  $dt \leftarrow 1$                                             ▷ Interval (s)
  $npkts \leftarrow 0$                                          ▷ Number of packets
  $meta\_set \leftarrow \{\}$
  for each $pkt \in capture$ do
      if $npkts$ is zero then
          $first\_ts \leftarrow pkt.sniff\_timestamp$
      end if
      $ks \leftarrow (pkt.sniff\_timestamp - first\_ts)/dt$
      if $pkt.ip.src \in nameservers$ then
          $meta\_set[ks][1] \leftarrow pkt.length$
      else
          $meta\_set[ks][0] \leftarrow pkt.length$
      end if
      $npkts \leftarrow npkts + 1$
  end for
  $time\_lst \leftarrow sorted(list(meta\_set.keys()))$
  $time \leftarrow range(time\_lst[0], time\_lst[1])$
  for each $t \in time$ do
      if $(kt := meta\_set.get(t)) \neq NULL$ then
          if $kt[0] \neq []$ then
              $client\_pps \leftarrow len(kt[0])$
              $client\_avgpktlen \leftarrow int(sum(kt[0])/len(kt[0]))$
          else
              $client\_pps \leftarrow 0$
              $client\_avgpktlen \leftarrow 0$
          end if
          if $kt[1] \neq []$ then
              $server\_pps \leftarrow len(kt[1])$
              $server\_avgpktlen \leftarrow int(sum(kt[1])/len(kt[1]))$
          else
              $server\_pps \leftarrow 0$
              $server\_avgpktlen \leftarrow 0$
          end if
          $datafile \leftarrow t, client\_pps, client\_avgpktlen, server\_pps, server\_avgpktlen$
      else
          $datafile \leftarrow t, 0, 0, 0, 0$
      end if
  end for
```

3.1.1.2 Emulating network behaviors of malware bots

The emulation of illicit DNS usages passes through using DNS Tunneling tools that embed data in DNS queries and that deliver DNS requests and responses between the client and the C&C server. The C&C server can afterwards forward the received data to the botmaster. There are a number of public DNS Tunneling tools with distinct implementations, using different DNS RR types, encoding schemes, and offering multiple levels of configurations. The three most common DNS Tunneling tools are [66], [67]:

- **Iodine**: has a wide range of possible configurations. It allows Base32, Base64, Base64u and Base128 encoding schemes, and several DNS RRs are supported, with `NULL` and `PRIVATE` being the ones that provide the faster upstream throughput. In addiction, Iodine uses EDNS0, a specification that allows the resolver to accept DNS messages of size larger than 512 bytes, the standard size of UDP messages proposed by RFC 1035.
- **DNS2TCP**: has limited configuration. DNS messages are encapsulated in `TXT` RRs and the encoding scheme used is the Base64. Performance wise, DNS2TCP has greater throughput comparing to Iodine, but higher overhead as well. A reason for the greater overhead is the frequency of *keep-alive* messages as DNS2TCP sends *keep-alives* every second. In Iodine, this parameter is configurable.
- **Dnscat2**: primarily designed to create an encrypted C&C channel over DNS. It supports a large number of RRs and encoding schemes, but has greater overhead and smaller throughput in comparison to the alternatives.

Once the C&C infraestruture is built up, there are two major approaches to deliver commands to malware bots: by either using *pull* mechanisms or *push* mechanisms [68], [69] (Figure 3.3). In *pull* mechanisms, bots retrieve commands actively from a C&C server, where botmasters publish them, following a publish–subscribe pattern, while in *push* mechanisms bots establish a session with C&C servers and passively wait for the arrival of commands. The former is typically used in centralized topologies, especially in HTTP botnets, while the latter is commonly used in P2P bots that relay commands and in IRC-based bots. In DNS botnets the study of these two types of C&C communications have been overlooked, thus one should beware of both implementations.

For the emulation of data exfiltration tasks, malware bots just need to exfiltrate the encoded data to DNS servers without being detected, by implementing the behaviors described later in The DNS threat landscape, and no activity is needed from the C&C server at all.

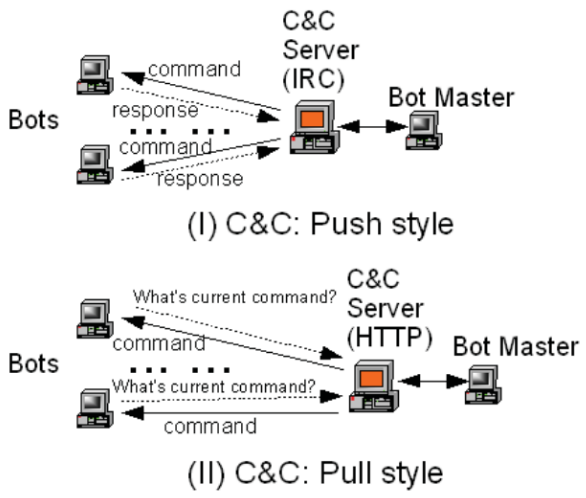


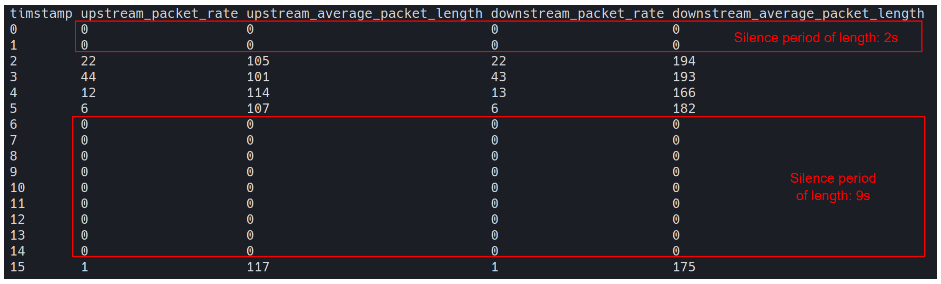
Figure 3.3: Two approaches for C&C communications. Source: BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic [70]

3.1.2 Feature extraction

The collected network traffic metrics may be numerical, but they have no profound insights, and do not show any meaningful attributes or relationships whatsoever [71]. Data are the must-have food for Machine Learning models, their quality is critical to their ability to make good predictions. If we feed those models with poor and meaningless data, they will not generalize well and improve from the given experience, that has no relevant representation in the first place.

Notwithstanding, network traffic metrics can be properly processed and used to extract patterns with real, underlying meaning. For instance, extracting statistical features, such as the mean and the standard deviation of packet lengths over a time period, from licit and illicit DNS traffic might reveal divergent distributions that could further indicate that a given machine is infected and being part of a botnet. This process of creating a reduced set of groups with meaningful representations from raw data is called feature extraction and it can speed-up the computations of AD algorithms while increasing their performance.

Having timestamped raw data also enables one to extract features that might reveal an underlying periodicity from malware traffic, such as activity and silence moments. In network traces, a silence moment is represented by a sampling slot with all metrics equal to zero, and an activity event by a sampling slot with at least one metric value other than zero. Sequences with one or more consecutive silence, or activity, moments are posteriorly grouped together in silence and activity periods, and their frequencies, as well as their lengths, can be measured. An illustration of two silence periods, one with a length of 2 seconds, and the other with a length of 9 seconds, can be seen in Figure 3.4.



```
timstamp upstream_packet_rate upstream_average_packet_length downstream_packet_rate downstream_average_packet_length
0        0                    0                              0                      0
1        0                    0                              0                      0
2        22                   105                            22                     194
3        44                   101                            43                     193
4        12                   114                            13                     166
5        6                    107                            6                      182
6        0                    0                              0                      0
7        0                    0                              0                      0
8        0                    0                              0                      0
9        0                    0                              0                      0
10       0                    0                              0                      0
11       0                    0                              0                      0
12       0                    0                              0                      0
13       0                    0                              0                      0
14       0                    0                              0                      0
15       1                    117                            1                      175
```

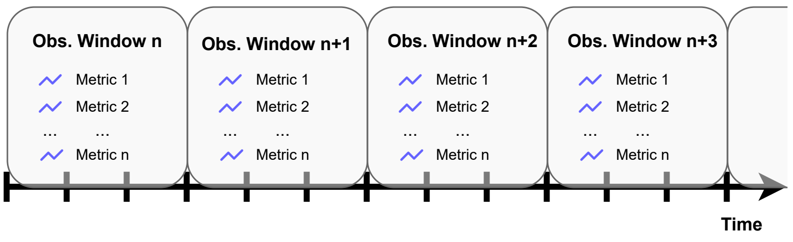
Figure 3.4: An example of silence periods in collected network metrics

3.1.2.1 Feature Extraction with Observation Windows

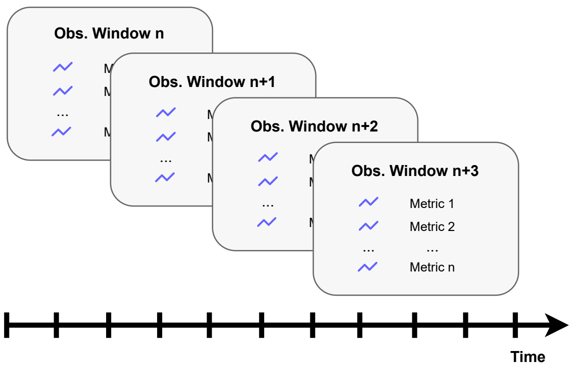
Observations windows is a technique that allows one to derive relevant features from a stream of metrics, or events, over consecutive time intervals [72], i.e., windows. The computed features describe events that occurred in a given timestamp, representing an observation, and multiple observations form a characterization of an entity, also called as profile, enabling one to group entities that show a similar behavior and to detect illicit entities that reveal behavioral deviations from the known group. A simple approach to observation windows, is to implement them in a sequential manner, where all windows have the same size θ and if a window is at sample i, then that window will end at sample $i + \theta$ and the next window would begin at sample $i + \theta + 1$ (Figure 3.5a). A disadvantage of this approach, however, is that, depending on the size of the window, different observation windows may extract very different patterns, being challenging to obtain a coherent representation of the underlying distributions.

A better way to apply observation windows to timestamped metrics is if the next window do not start at a steeply step such as $i + \theta + 1$, but rather at a lighter step, such as $i + \Delta$, with $\Delta < \theta$ (Figure 3.5b). In this approach, closer windows will be partially overlapped, allowing for longer periods of observation and a smoother relationship between data points. Another advantage of having partially overlapped windows (sliding windows) is that, although the behavioral differences between samples will be smaller, creating similar observations for small sliding values, they generate significantly more samples than the sequential approach, wherefore Machine Learning models have more data to learn from, increasing the odds of a better generalization, and they also allow for longer periods of observation, while maintaining a short period of decision.

Another common window technique for a robust and rich to feature extraction is to use sliding windows with multiple observation sizes, creating different meaningful representations and perceptions about the collected data by varying the observation time interval. The dataset must be split according to the larger window size, as well as the specified sliding value, and for every sub-window at sample i the relevant features



(a) Sequential observation windows approach



(b) Sliding observation windows approach

Figure 3.5: Comparison between sequential observation windows and sliding observation windows over timestamped raw data

are computed. In Network-based botnet detection [73], [74], a common way of extracting relevant features is to extract descriptive statistics, of which seek to describe, i.e. give meaning to, the collected data by giving short summaries about the sample metrics in a sub-window. Furthermore, descriptive statistics can be divided into measures of central tendency and measures of variability: Measures of central tendency describe the center position of a probability distribution for the sample metrics, for instance, by measuring the frequency of each data point in the distribution and describing it using the mean, median, mode, or other central tendency measurement that gives an insight about their common patterns, while measures of variability, analyze the dispersion and the shape of the distribution, describing how the data is distributed within the sub-window, and common measures of dispersion are the variance, standard deviation and quantiles. With regard to heavy-tailed distributions (distributions that goes to zero slower than ones with exponential tails), one can also use trimmed estimators, such as, the trimmed mean and the interquartile range, of which are considered more robust to the presence of extreme values in data, specially when dealing with mixture distributions and distributions that have longer tails than the assumed normal distribution [75]. As showcased in Figure 3.6, the mean, 5% trimmed mean, standard deviation, 85, 90, 95 and 99 quantile, and maximum values with regard to the collected traffic metrics and the activity and silence periods are computed. For the latter, also the number of silence and activity periods are retained. These statistics captured from different sub-windows are then joined into a single feature vector, forming the dataset.

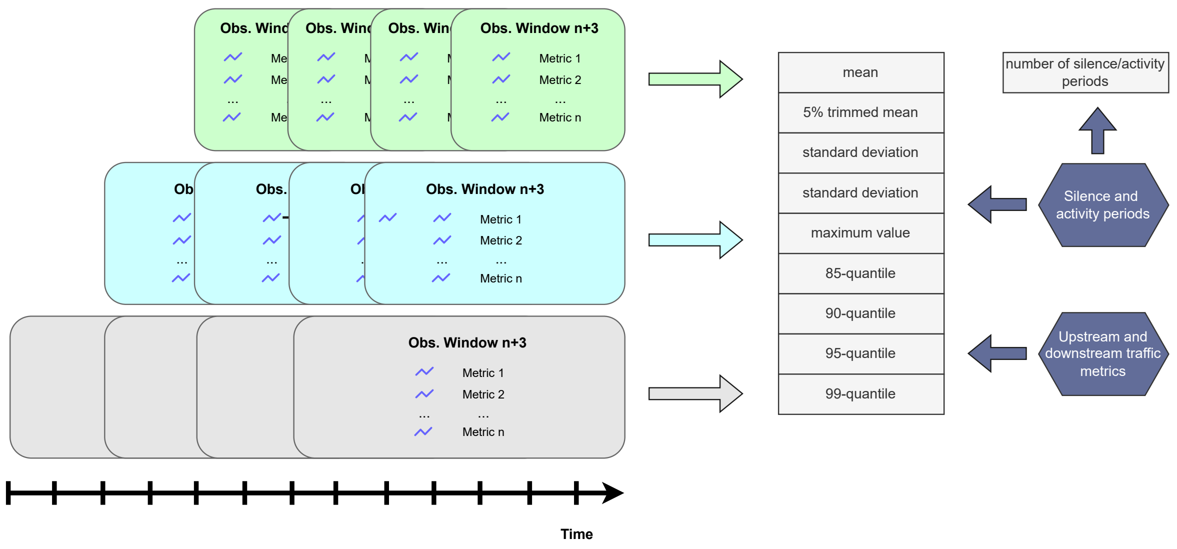


Figure 3.6: Diagram of features extracted from multiple sub-windows.

3.1.3 Data scaling

Data preprocessing is an important step to achieve good outcomes in novelty detection tasks. It includes removing possible outliers and noise from the data, dealing with incomplete or missing values, and transformation of data into a common scale. Because our data was produced in controlled environments, they should be outlier and noise-free, however, as features came from multiple metrics, they are in different ranges. Most predictive models do not generalize well on features whose scales differ vastly, therefore their ranges should be either scaled or transformed so that each feature contributes equally to the final predictive result. Having scaled variables also help Machine Learning model that have gradient descent as an optimization technique to converge much faster, and coefficients are penalized properly when regularization techniques are applied in loss functions.

The two most common feature scaling methods are Normalization and Standardization. In Normalization, data are transformed into a homogeneous range of values, often between 0 and 1, or -1 and 1, it is typically used when the underlying distribution unknown, or when the data do not follow a bell-shaped curve, and it is computed as follows:

$$x' = \frac{x - min}{max - min} \tag{3.1}$$

While in Feature Standardization, the values of each feature are set with a zero-mean and a standard deviation equals to one, being calculated by the formula:

$$x' = \frac{x - \overline{x}}{\sigma} \tag{3.2}$$

where σ is the standard deviation, of the feature vector, and $\overline{x}$ the mean value of the feature vector. Standardization normally works best when data follows a normal distribution, although this is not always true, and so, one should test and see what scaling technique is more appropriate in that particular case.

After choosing the scaling technique, the mean and standard deviation of features, with regard to Standarization, or the maximum and minimum values of features, with regard to Normalization, should be computed. These computations **must be done only with the training set**, while the transformation is applied to both the training and test sets.

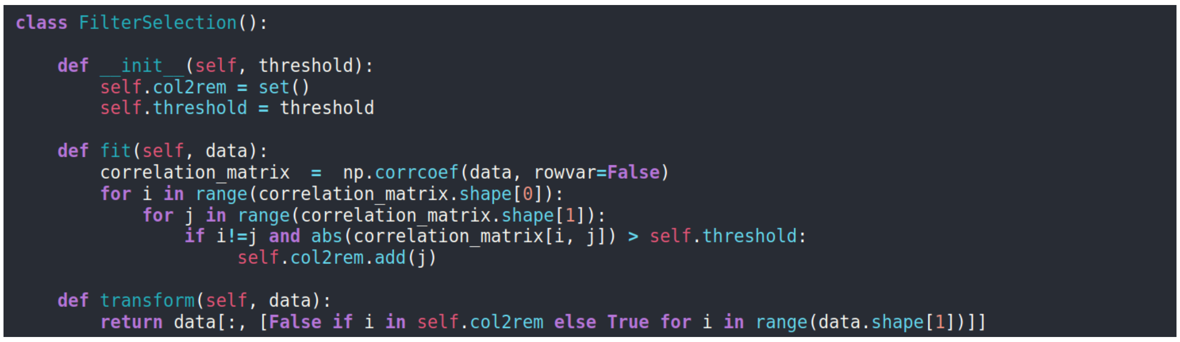
3.1.4 Dimensionality Reduction

Although data scaling guarantees that features have an equal contribution, it does not imply that these features are equally important for the decision process of the model. As an example, if we consider that there are n sub-windows in each observation window, with m extracted features in each sub-window, then, each observation will have $n \times m$ features. Despite the fact that one should extract as many relevant features as possible, in favor of a detailed and accurate representation of network traffic behaviors, most likely, some extracted features may be irrelevant, and/or redundant, to the class value. Furthermore, having many features but a low number of samples may decrease the performance of Machine Learning models. This phenomena is called the Curse of Dimensionality, and it was discussed in Dimensionality Reduction.

In this situation, one could thing that using Feature Selection methods would resolve this problem by using only a subset of those features to train the AD models. However, those methods require the presence of both licit and illicit class values for the filtering of irrelevant features, and since the objective is to design an AD system, illicit data should not be used during the training phase, in order to build a reliable representation of normality. To avoid having the models biased towards known malware bots traffic patterns, one should consider techniques that do not require the use of the target value, resorting in unsupervised dimensionality reduction methods.

The most common unsupervised dimensionality reduction technique is the PCA, which transforms a large set of features into a smaller set of uncorrelated features, while preserving the most variance in the dataset. This technique was already discussed in Principal Component Analysis, nevertheless, other unsupervised dimensionality reduction techniques, such as the High Correlation Filter, can be used. The High Correlation Filter uses the Pearson correlation matrix to identify correlations between features. Once they are calculated, one can set a threshold to remove the features that show a correlation higher than the threshold. This algorithm can be seen in Figure 3.7.

In a similar way to data scaling, the eigen decomposition with regard to PCA, and the features to remove with regard to the High Correlation Filter must be computed only with the training set, while the transformation is applied to both the training and test sets.



```python
class FilterSelection():

    def __init__(self, threshold):
        self.col2rem = set()
        self.threshold = threshold

    def fit(self, data):
        correlation_matrix  =  np.corrcoef(data, rowvar=False)
        for i in range(correlation_matrix.shape[0]):
            for j in range(correlation_matrix.shape[1]):
                if i!=j and abs(correlation_matrix[i, j]) > self.threshold:
                    self.col2rem.add(j)

    def transform(self, data):
        return data[:, [False if i in self.col2rem else True for i in range(data.shape[1])]]
```

Figure 3.7: High Correlation Filter Algorithm

3.1.5 The Learning Process and model evaluation

Once the data is preprocessed, it is finally ready to be used by AD models. First, we need to split the dataset with licit instances into training and test sets. The main reason to have a test dataset is to test the model against new and unseen malware bot traffic, therefore the test set must contain, not only licit samples, but also samples from new and unseen bot malware attacks.

When evaluating different hyper-parameter settings for models using a single test set, there is risk of overfitting on the test set, as parameters can be tweaked until the model fits optimally on that particular set, not generalizing well on a real-world scenario. A straightforward solution for this problem is to create a new set, called validation set, also with licit and illicit instances, to be used for hyper-parameter tuning, and once the experiment is done, a final evaluation is done on the test set. However, having three sets can impact greatly in the number of samples used to train the model and build a good representation of normality, and the results can depend on a particular random choice of instances in the training and validation sets.

Therefore, a better practice for classic Machine Learning techniques is to use Cross-Validation techniques, such as the k-fold Cross-Validation depicted in Figure 3.8, that use different partitions of the data, in a iterative way, to train the model and choose the best hyper-parameter settings for more accurate out-of-sample and overfitting estimations, specially when data is scarce, or the number of features is large. The reported performance metrics in the loop are then averaged to find the best hyper-parameters and a final evaluation is done using the test set. Furthermore, one can nest the Cross-Validation to simultaneously select the best set of hyper-parameters and evaluate them on different test sets, multiple times. The two ways of doing a nested Cross-Validation are:

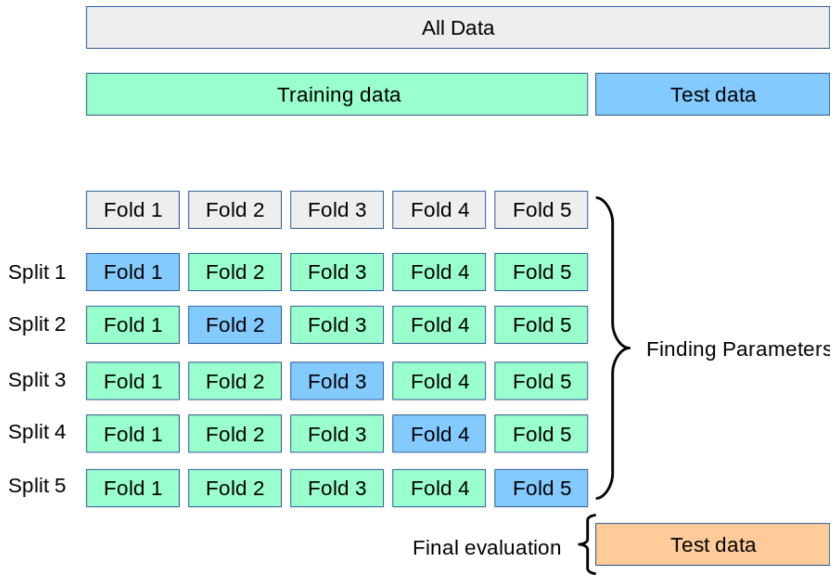


Figure 3.8: k-fold Cross-validation. Source: Scikit-Learn

- ***k*l*-fold Cross-Validation**: The total dataset is split into k sets. The $k-1$ sets are combined to form a training outer set and the remainder is the test set, used to produce a unbiased model evaluation. This procedure is repeated for each of the k sets. Each outer training set is further sub-divided into l inner sets. A set is selected as a inner validation set for model hyper-parameter tuning and the other $l-1$ inner sets are used as training sets, and it is also repeated for each one of the l sets. After choosing the best hyper-parameter settings for the inner Cross-Validation, a new model with the best hyper-parameters is fit on the entire outer training set and evaluated with the outer test set. This approach is computationally expensive as it performs $k * l$ iterations at least, but it should return accurate estimates when choosing the best hyper-parameter settings.
- ***k*-fold Cross-Validation with validation and test set**: A single k-fold Cross-Validation is used for both validation and testing purposes. The total dataset is split into k sets. The $k-1$ sets are combined to form the training set, and the remaining set is used to form the validation and test sets, to further mix with malware bot samples. Similar to k*l-fold Cross-Validation, the training set is used to construct a representation of normality, the validation set for the hyper-parameter tuning, and the test set to evaluate the model with the best hyper-parameter settings.

In both approaches, the validation and test sets should have an equal amount of licit and malware bot samples, in order to not get skewed performance metrics. They are both implemented, however, due to the intensive computational demand for the former, the latter was chosen to test our models, and the splitting process at fold k is illustrated in Figure 3.9.

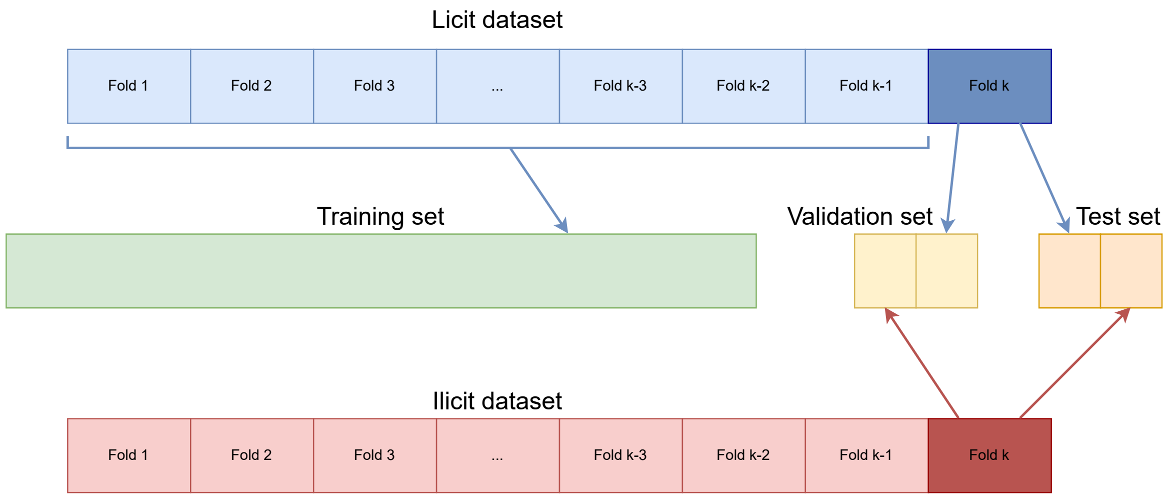


Figure 3.9: Illustration for the k-fold Cross-Validation with validation and test set at iteration k

At each fold, the k-fold Cross-Validation with validation and test set approach selects the hyper-parameter setting that obtained the best *weighted F1-Score*, and an additional evaluation is made with the test set, obtaining the *weighted F1-Score* and the False Positive Rate (FPR). Consequently, the precision and recall metrics must be computed from the confusion matrix for both normal and abnormal classes. If we consider that the known observations are in group i, the predicted values in group j, 0 is the class number of licit samples, and 1 the class number of anomalies, then we get the following equations:

$$precision_i = \frac{C_{ii}}{\sum_j C_{ji}} \tag{3.3a}$$

$$recall_i = \frac{C_{ii}}{\sum_i C_{ji}} \tag{3.3b}$$

$$F1_i = \frac{2 \cdot precision_i \cdot recall_i}{precision_i + recall_i} \tag{3.3c}$$

$$F1_{weighted} = w_0 \cdot F1_0 + w_1 \cdot F1_1 \tag{3.3d}$$

$$FPR = 1 - precision_0 \tag{3.3e}$$

where, in this case, 0.50 is the value of w_0 and w_1.

To choose between different estimators, the results of weighted F1-Scores and FPRs must be averaged and a confidence interval must be computed according to the equation 3.4, where σ is the standard deviation of a set of given metric values, n its total number, and Z, a tabular value, is the $\frac{1-C}{2}$ percentile point of the standard normal distribution for a confidence interval of $C\%$.

$$CI_C = Z\frac{\sigma}{\sqrt{n}} \tag{3.4}$$

Once the best estimator is chosen, the best hyper-parameter setting is selected according to the best weighted F1-Score and FPR.

3.2 Summary

The methodologies presented throughout this chapter aim to establish a robust and privacy-focused framework for Network-based bot malware detection over DNS, of which has been largely overlooked in the literature. The majority of strategies use DPI techniques and unnecessary PII. With this in mind, the main features of these methodologies are:

- Privacy-focused solutions for bot botnet and bot malware detection: since the arrival of new privacy and data protection regulations, invasive strategies can leave companies and organizations *dipped* in privacy related issues, loosing the trust of their costumers.
- It focuses on pattern and behavioral analysis, since analysing the patterns of packet lengths and packet rates to analysing the behaviors of silence and activity periods.
- It does not need to feed the models with every imaginable type of anomalous behavior, as they only train with "clean" data, building a representation of normality.
- It resorts in multiple feature engineering techniques to avoid feeding some models with non-relevant data.
- It focuses in optimizing the model performance based on low FPR and high *weighted F1 Score*, penalizing greatly the incorrectly classified cases.

By using Novelty Detection models, we are in a better position to cover a wide range of out-of-normal behaviors, than we would if we used classic binary classifiers, yet we also need anomalous samples to evaluate the performance of such models. These mechanisms will be further discussed in chapter 4, where they will be employed to perform novelty detection in different scenarios, using different DNS tunneling frameworks, C&C methods and behaviors.

CHAPTER 4

Methodologies evaluation and Results

This chapter describes the conducted experiments to validate the methodologies presented in Chapter 3 for AD, including the realization of different scenarios that misuse the DNS, since the establishment of C&C infrastructures to data exfiltration. The goal is to evaluate the accuracy of the proposed methodologies, identifying correctly malware activity over DNS while having low False Positives.

4.1 Licit traffic collection

As stated in section Data collection, Novelty Detection models need a good variety of possible licit usages of DNS, in order to successfully construct a model of normality. Furthermore, the licit DNS activity must be collected under a controlled environment to ensure the integrity of the *ground-truth*.

With that in mind, the licit DNS traffic was collected using Wireshark [1], while performing multiple activities in a web browser, such as reading the news, studying, using social networks and consuming media, and streaming music via Spotify's own application [2], for, around, 3 hours straight. These use cases were produced under the three major operating systems: Linux Ubuntu [3], macOS Big Sur [4] and Windows 11 [5]. In some Linux Ubuntu traces the option to send diagnostic data is activated, while in others the machine do not send any diagnostic data whatsoever. In Windows and macOS systems this option is required. For each operating system, the browser used

[1]shttps://www.wireshark.org/
[2]https://www.spotify.com/pt-pt/
[3]https://ubuntu.com/download
[4]https://apps.apple.com/us/app/macos-big-sur/id1526878132
[5]https://www.microsoft.com/pt-pt/software-download/windows11

in captures was the pre-installed one, namely Firefox [6] in Linux Ubuntu, Safari [7] in macOS and Microsoft Edge [8] in Windows. In order to cover a wide variety of licit use cases, some of them use ad blockers, while others don't, and in one use case, even the JavaScript was disabled and media larger than 50kb was blocked. To prevent false alarms from encrypted, but licit traffic, two use cases in Linux Ubuntu using DNS over TLS (DoT), one with the ad blocker active and other without using ad blockers, were also captured.

The collected traces were afterwards transformed in traffic metrics, using the Algorithm 1, and examples of upstream and downstream throughput in a Time Series and Histogram graph, as well as the upstream and downstream average packet length Histogram in licit traffic can be consulted in Figures 1, 2 and 3, respectively.

4.2 Emulation of malware bot attacks

AD algorithms may be trained only with one class, but they must be tested with new and unseen normal observations, as well as with new and unseen abnormal data points, that are, for our case, observations extracted from emulated botnet activity.For this reason, we partially simulated a corporate network using GNS3 with an infected computer, a DNS Server and a Router, a public DNS server used by the botmaster to register a domain for the further establishment of the covert channel, a C&C server and the botmaster's computer. The DNS servers were implemented with BIND9, using the default settings, and static routing was used at the core of the simulation. All devices are accessible via SSH using a NAT interface, and they can also be connected to real world Internet using a TAP interface. The simulated environment is depicted in Figure 4.1. As a side note, since we did not configure any mapping methods between private and public corporate addresses, all nodes in the Internet simulation know the company's private addresses, although, this is not a big issue as we are only interested in monitoring the link between the infected machine and its corporate DNS server.

Once the virtual network simulation is set up, now we can think about how to emulate botnet traffic for different possible scenarios and what technologies to use. We decided to pick the two most common DNS tunneling tools to emulate botnet traffic: **Iodine** [9] and **DNS2TCP** [10]. In Iodine, we decided to use the `NULL` RR type as it provides the faster upstream throughput among the available RRs, and we configured the `keep-alive` messages to have a low enough frequency to not get `SERVFAIL` messages, i.e. 9 seconds, while in DNS2TCP the RR used is the `TXT` query type with Base64.

[6]https://www.mozilla.org/pt-PT/firefox/new/

[7]https://www.apple.com/safari/

[8]https://www.microsoft.com/pt-pt/edge?form=MA13FJ

[9]https://github.com/yarrick/iodine

[10]https://github.com/denisix/dns2tcp

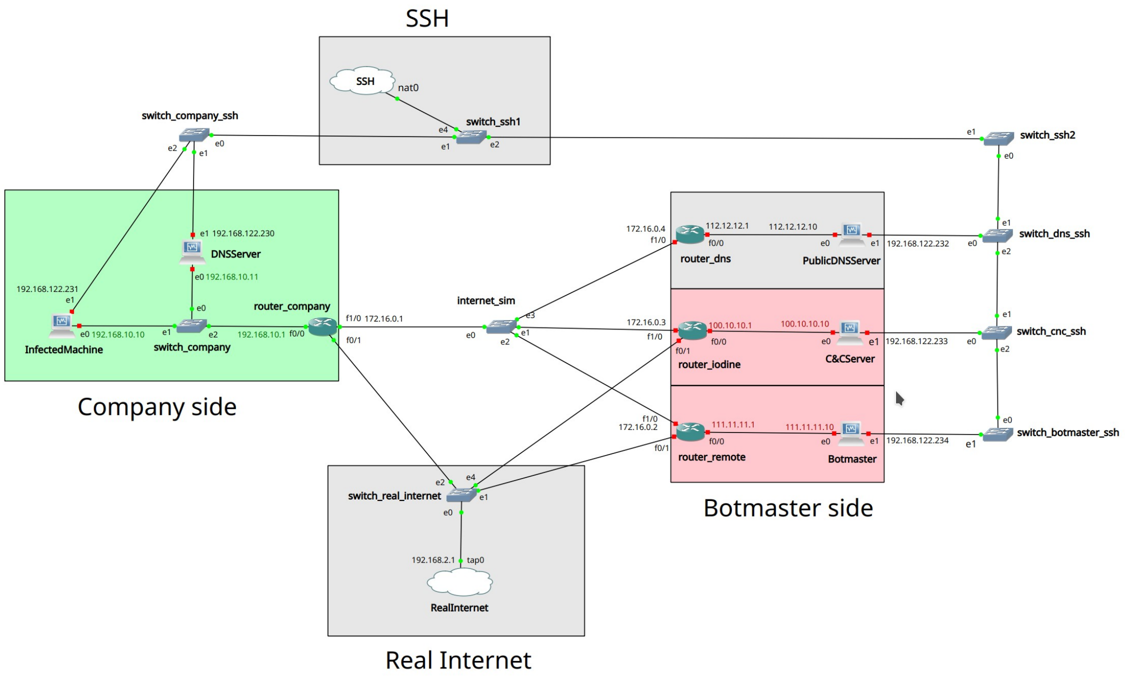


Figure 4.1: Simulated corporate environment for the emulation of botnet attacks

To have a broad diversity of possible attacks, 6 scenarios were a botnet can try to emulate a normal behavior, or to blend with normal traffic without being detected, were produced: the Scenario 1, were we emulate a C&C channel over DNS using the original Iodine and DNS2TCP tools, the Scenario 2, were we emulate exfiltration of data, also with the original Iodine and DNS2TCP tools, two more advanced scenarios, Scenario 3 and Scenario 4, were we emulate a C&C channel and exfiltration of data with a modified version of the DNS2TCP tool (the motivation behind this decision will be discussed in that subsection), the Scenario 5, were we emulate stealth C&C traffic with Pull mechanisms, and finally the Scenario 6, were we emulate a slow and low exfiltration of data using the modified version of the DNS2TCP. The algorithm 1 was again used to extract the raw features of traffic generated by the botnet emulation in all scenarios.

As referenced in section The DNS threat landscape, *keep-alive* messages are useful for botmasters to know if machines are still under their control or not. Nonetheless, sophisticated botnets can use more adequate methods to check the status of their bots, or not use any methods at all, and thus one should also train the developed AD algorithms without those messages. In Iodine, *keep-alive* subdomains always have 8 characters in length and they all can be decoded without `UnicodeDecodeError` exceptions, while in DNS2TCP *keep-alive* subdomains always have 10 characters in length. In both tools any encoded data that is passed through has a larger size than their *keep-alive* messages, consequently, two scripts to remove those messages were created: the Algorithm 2 and the Algorithm 3.

Algorithm 2 Remove DNS2TCP *keep-alive* messages

```
Require: $capture, outputfile\_name$
  $pktdump \leftarrow PcapWriter(outputfile\_name, append = True)$
  for each $pkt \in capture$ do
      $in\_pkt = NULL$
      if $"DNS" \in pkt$ then
          $name\_query \leftarrow pkt["DNS"].qd.qname.decode()$
          $names \leftarrow name\_query.split('.')$
          if $len(names[0]) = 10$ and $(pkt.len = 65$ or $pkt.len = 90)$ then
              $nop$
          else
              $in\_pkt = pkt$
          end if
      end if
      if $in\_pkt \neq NULL$ then
          $pktdump.write(in\_pkt)$
      end if
  end for
```

Algorithm 3 Remove Iodine *keep-alive* messages

```
Require: $capture$, $outputfile\_name$
  $pktdump \leftarrow PcapWriter(outputfile\_name, append = True)$
  for each $pkt \in capture$ do
      $in\_pkt = NULL$
      if $"DNS" \in pkt$ then
          try
              $name\_query \leftarrow pkt["DNS"].qd.qname.decode()$
              if $(".t.b0t.pt" \in name\_query)$ and $(len(name\_query.split(".")[0]) = 8)$
  then
                  $nop$
              else
                  $in\_pkt \leftarrow pkt$
              end if
          catch UnicodeDecodeError
              $in\_pkt \leftarrow pkt$
          end try
      end if
      if $in\_pkt \neq NULL$ then
          $pktdump.write(in\_pkt)$
      end if
  end for
```

4.2.1 Scenario 1: C&C over DNS with the Push mechanism using standard behaviors

This scenario aims to address the C&C infrastructures that use Push mechanisms to run commands on the underlying bots. The control of infected machines can be done using simple remote shell programs, like Ncat, to more complex and robust C&C frameworks, like Meterpreter or Merlin, of which allow botmasters to establish stealth channels that mimic licit network traffic. For these reasons, we produced C&C traffic using Ncat [11] to send commands and receive data from the infected host over DNS2TCP and Iodine DNS tunneling tools, and we also produced Ncat traffic encrypted with SSL.

Most C&C frameworks do not support the DNS protocol, and their ability to produce stealth traffic over DNS tunneling software is poor, therefore a script in Python [12] was made having in mind that the attacker can choose different probability distributions that resemble licit behaviors, for instance, a beta negative binomial distribution for the throughput and a normal distribution for the packet length emulation, while trying to guess their parameters. Because sometimes relying in parametric distributions may not be the best option to reproduce licit behaviors, some network traces were produced

[11]https://nmap.org/ncat/

[12]https://www.python.org/downloads/release/python-380/

with additional workarounds, such as adding a weighted choice to samples drawn from those distributions. Examples of generated throughput and average packet length for this scenario can be seen in Figures 4 and 5.

4.2.2 Scenario 2: Data exfiltration over DNS using standard behaviors

As seen previously in section More advanced topologies, botnets may not be listening to commands during all of their life-cycle, but rather be in a different stage. One of the most critic stages of an attack is exfiltration of data, and during this phase, botnets can steal private and sensitive data without any activity from the botmaster or their C&C servers whatsoever.

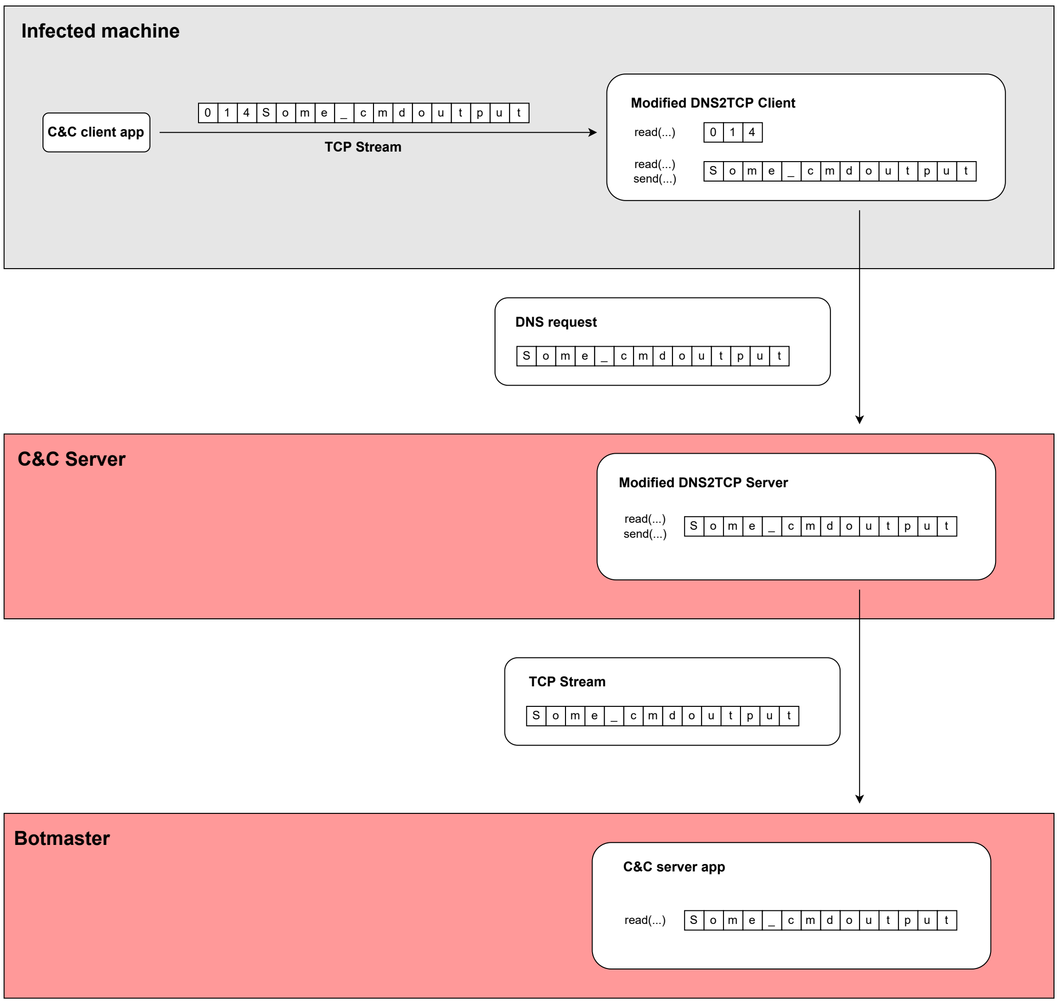
In this scenario, we developed a script in Python to exfiltrate a file over Iodine and DNS2TCP while trying to emulate the normal activity using again different probability distributions that resemble licit packet length and throughput patterns of a targeted machine, in order to evade being detected.

4.2.3 Scenario 3: C&C with the Push Mechanism over DNS that mimics licit behaviors

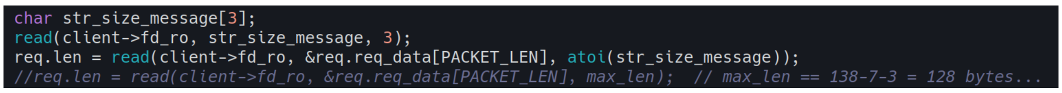
Using available remote shells and C&C frameworks can provide low latency and robustness to one's botnet, however their patterns and behaviors are well known to cyber-security blue teams. Moreover, using DNS tunneling software, like Iodine and DNS2TCP, creates a difficult challenge to emulate licit packet lengths, as this software relies in a pre-fixed buffer size to send data, and, although TCP provides a reliable way of delivering messages, being stream oriented means that there is no assuredness about the preservation of message bounds.

These characteristics about the TCP protocol and DNS tunneling software led us to create a new script to send C&C commands and get the output data, written in C, where the the Nagle's Algorithm, responsible to reduce the number of small packets sent over the network with the goal of improving network congestion, was disabled with the socket option `TCP_NODELAY`, and `TCP_QUICKACK` was enabled to stop the delay of TCP acknowledgments that, one again, initially served to improve network congestion. It is important to consider that since we control the throughput, the silence periods and the packet lengths, these features are not needed and can be inconvenient for a botnet that attempts to emulate a normal behavior.

The last thing that we need is a way to ensure that messages are delivered with the size that we want. This can be done at the application level, by changing the source code of DNS tunneling tools. We picked the DNS2TCP source code to be edited, since it can send more packets per second (pps) than Iodine, thus being more capable of generating a realistic throughput. In our client and server scripts, messages are sent with the message size attached at the beginning, occupying 3 octets, i.e. the packet size can be up to 999 bytes, being more than enough to reproduce realistic packet sizes. In the modified DNS2TCP [13], client and server applications receive those messages, read the first 3 octets and store the number in a variable called `str_size_message`, and then, read up to `str_size_message` bytes from the socket's file descriptor. This implementation is illustrated in Figure 4.2, and now we have a reliable way of tunneling packets with a specific size.



(a) Visual Example of a command output sent to the botmaster



```
char str_size_message[3];
read(client->fd_ro, str_size_message, 3);
req.len = read(client->fd_ro, &req.req_data[PACKET_LEN], atoi(str_size_message));
//req.len = read(client->fd_ro, &req.req_data[PACKET_LEN], max_len);  // max_len == 138-7-3 = 128 bytes...
```

(b) Modified code

Figure 4.2: Changes made in the client application of DNS2TCP

Comparing to the techniques used in Scenario 1 for the emulation of normal behaviors, we use a different method to send data back to the C&C server, instead of estimating

[13]https://github.com/brnaguiar/dns2tcp-custom

parameters from a parametric distribution. In the server application, commands are sent randomly between 0 and 15 seconds, and the client application receives them, runs them and returns their outputs according to empirical mass functions extracted from observed licit behaviors of the target machine. For instance, a *bot binary* might observe the packet rate, packet lengths and silence periods of the host machine to later generate botnet traffic according to the empirical probabilities computed from those characteristics. Furthermore, we applied the Laplace Smoothing technique to smooth the empirical data, supposing that the attacker might want also to generate *similar* distributions to the licit ones, but not *equal.* This method is given by the formula:

$$P_i = \frac{x_i + \alpha}{N + \alpha d} \tag{4.1}$$

Where x_i/N is the empirical probability, α the smoothing parameter, and d the number of measured variables. This technique is present only in few captures, with small smoothing values to ensure that the malware behavior doesn't deviate that much from the licit distribution. A comparison between licit and illicit traffic generated by the modified DNS2TCP is portrayed in Figures 6 and 7. It is worth noting that the generated packet sizes may not be identical to the licit ones because of the DNS2TCP encoding scheme.

4.2.4 Scenario 4: Data exfiltration over DNS that mimics licit behaviors mechanisms

In this scenario, we emulate a exfiltration of data with a similar technique applied to Scenario 2, using the modified DNS2TCP. A script to exfiltrate a file made in C with the Nagle's Algorithm deactivated and the `TCP_QUICKACK` enabled mimics a licit behavior according to empirical probabilities extracted from observing licit DNS activity of the infected machine.

4.2.5 Scenario 5: C&C with the Pull Mechanism over DNS mixed with licit traffic

In C&C infrastructures that use Pull mechanisms, bots usually ask for commands, instead of waiting for them. This approach is more loose in comparison to the Push mechanism, meaning that commands have considerably more latency to arrive to the bot end, although the bot end usually generate less traffic as it typically does not send the command's output back to the botmaster. A small API was developed in Python with FastAPI [14] to replicate the behaviors of bots that request new commands and the responses from the server. Attempting to evade detection by generating little traffic, requests are made within an interval between 10 and 15 minutes. An example of generated requests and responses is depicted in Figure 4.3.

[14]https://fastapi.tiangolo.com/

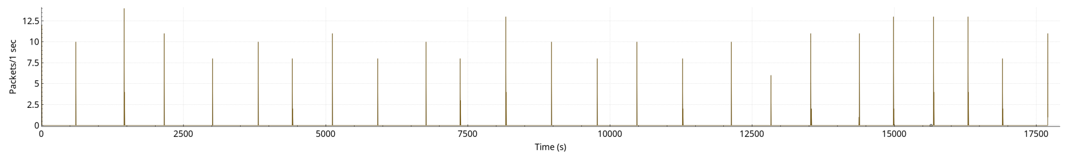


Figure 4.3: Example of illicit API requests and responses DNS activity

Because one could figure out promptly that it is not a normal behavior when comparing it to the licit DNS traffic captured in Section Licit traffic collection, this simulated botnet activity was later mixed with the licit traffic, in a way that it becomes really difficult to detect the abnormal activity, as showcased in Figure 4.4 for the example of Figure 4.3.

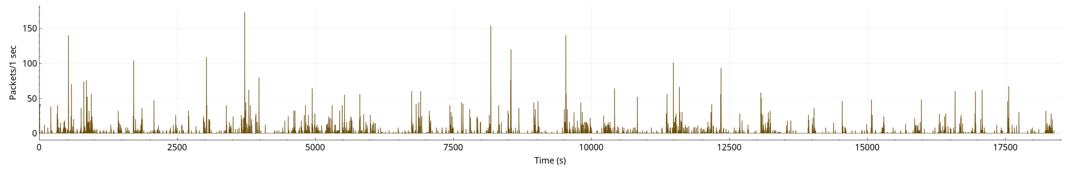


Figure 4.4: Example of illicit API requests and responses mixed with licit DNS activity

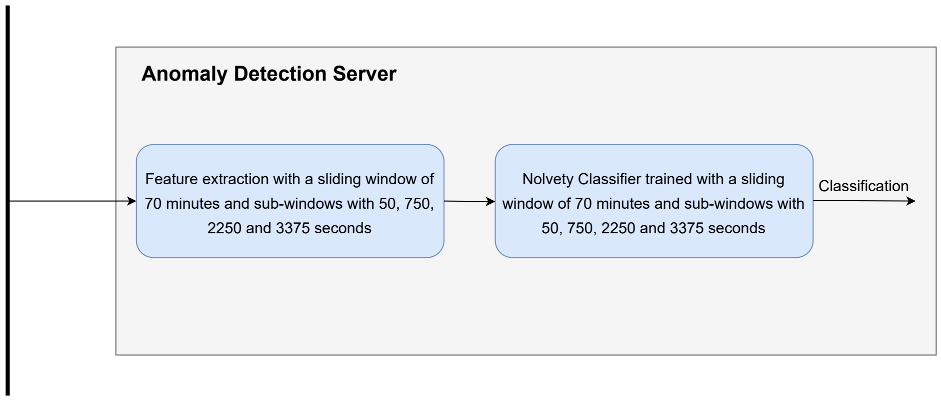
4.2.6 Scenario 6: Low throughput Data exfiltration over DNS mixed with licit traffic

Sophisticated botnets may generate low throughput traffic when exfiltrating data, aiming to be practically unnoticeable to security measures and impossible to distinguish from normal traffic. For this case, we used the modified DNS2TCP and applied the same techniques employed in Scenario 3 and Scenario 4 to generate packet lengths and throughput, although the latter was restricted to a maximum of 4 pps. To emulate slow DNS activity, we decided to code a weighted list were the script samples a random number and, based on it, *sleeps* between 6 and 10 seconds with a probability of 0.20, for 7 minutes with a probability of 0.20 and between 11 and 15 minutes with a probability of 0.60.

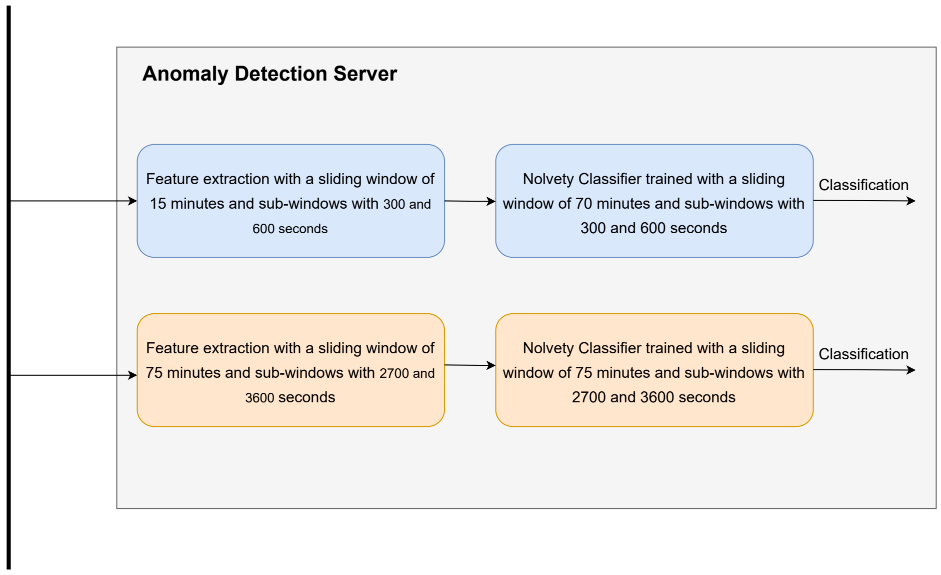
Similar to Scenario 5, this malware activity was later mixed with licit traffic, to be challenging for AD algorithms to detect the presence of illicit behaviors.

4.3 Dataset Exploration

All datasets were produced with the sliding windows approach, with multiple observation sizes, and for each scenario, a dataset has around 30 hours worth of data. When in production, the trained algorithms may live in a Anomaly Detection Server (ADS), and this ADS may have multiple instances of AD models trained with one or multiple sliding window sizes, therefore, models trained with smaller window sizes do not need to wait for models trained with bigger observation sizes to do the traffic classification task, while still having more moments of decision. Because of these reasons, we decided to create datasets with different observation window sizes, namely datasets with a sliding window of 15 minutes and sub-windows of 300 and 600 seconds, datasets with a sliding window of 75 minutes and sub-windows of 2700 and 3600 seconds, and datasets with a sliding window of 70 minutes and sub-windows with 450, 750, 2250 and 3375 seconds, all with a 2-second sliding distance. The last sliding window approach was produced to cover every anomaly that might occur when only one instance is presented in a ADS, and the first two sliding window approaches were produced with the purpose of having two AD instances with different sliding window sizes running in the ADS. The Figure 4.5 exemplifies the proposal of having multiple instances trained with multiple sliding observation window sizes.


Anomaly Detection Server
Feature extraction with a sliding window of 70 minutes and sub-windows with 50, 750, 2250 and 3375 seconds
Nolvety Classifier trained with a sliding window of 70 minutes and sub-windows with 50, 750, 2250 and 3375 seconds
Classification

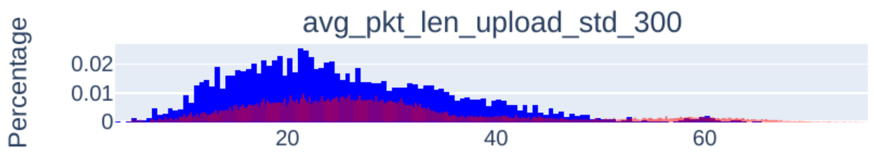
(a) ADS with one instance trained with a dataset with a sliding window of 70 minutes


Anomaly Detection Server
Feature extraction with a sliding window of 15 minutes and sub-windows with 300 and 600 seconds
Nolvety Classifier trained with a sliding window of 70 minutes and sub-windows with 300 and 600 seconds
Classification
Feature extraction with a sliding window of 75 minutes and sub-windows with 2700 and 3600 seconds
Nolvety Classifier trained with a sliding window of 75 minutes and sub-windows with 2700 and 3600 seconds
Classification

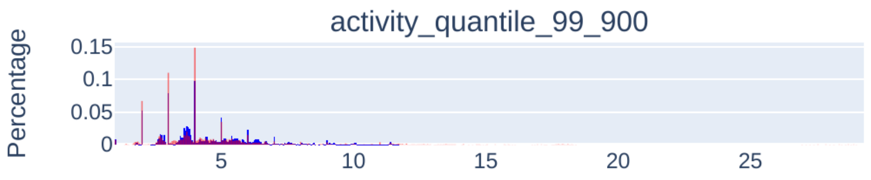
(b) ADS with two instances trained with datasets with sliding windows of 15 and 75 minutes.

Figure 4.5: Two examples of a possible ADS

A comparison between all licit (in blue) and illicit (in red) data points from all scenarios, including features extracted from traffic with *keep-alive* messages, can be seen in appendices 5.1, 5.1 and 5.1. When analyzing these graphs, and comparing the features `avg_pkt_len_upload_std_300` and `activity_quantile_99_900` from the Dataset with a sliding window of 15 minutes with the same features without *keep-alive* messages showcased in Figure 4.6 from the same dataset, one could that see that malware bots that send *keep-alive* messages often produce traffic patterns that are very divergent and distinguishable from licit behaviors, as *keep-alive* messages create many outliers in data, whereas illicit traffic without them is much more blended in licit instances. These *keep-alive* messages are present only in datasets that use standard behaviors, as periodic messages change the behavioral patterns of bot malware traffic when mimicking licit behaviors and their frequency change the low throughput nature of Scenarios 5 and 6.



(a) A comparison between licit (in blue) and illicit (in red) data points without *keep-alive* messages for the feature Average Packet Length Upload - Standard Deviation - of a 300 second observation window

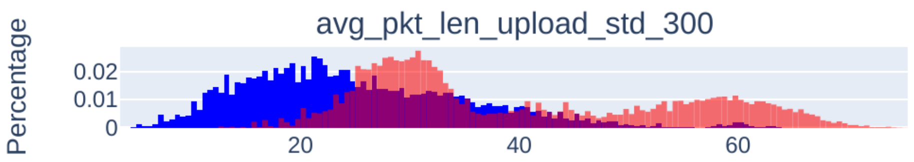


(b) A comparison between licit (in blue) and illicit (in red) data points without *keep-alive messages* for the feature Activity - 99-quantile - of a 900 second observation window

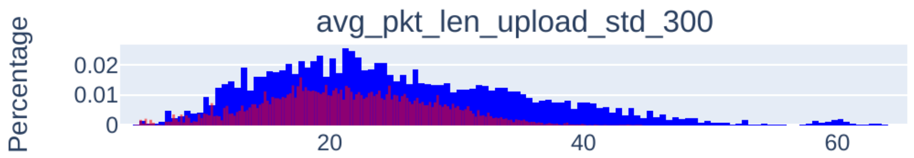
Figure 4.6: Example of features without *keep-alive* messages

Furthermore, if we compare illicit traffic emulated with different behavioral approaches (Figure 4.7), such as, malware bots that use standard behaviors, malware bots that mimic licit traffic and malware bots that mix with licit traffic, we will see that using standard behaviors, i.e. using parametric models, often produce traffic with more divergent distributions and that low throughput malware traffic when mixed with licit traffic can be very effective in deluding the current security systems.

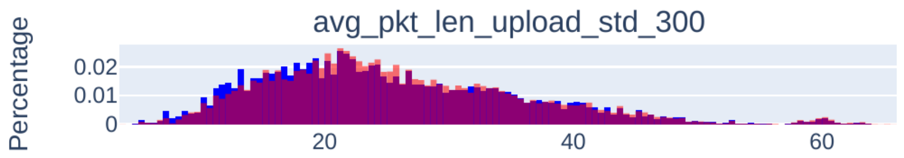
If we display the data into Scatter plots, we will see that it is possible for licit data to be linearly separable from illicit data with keep-alive messages, for large enough observation window sizes. Figure 4.8 showcases some combination of features that make the licit and illicit data instances linearly separable. For instance, the data from features `pps_upload_max_2250` and `pps_upload_quantile_85_2250` for the Dataset



(a) A comparison between licit (in blue) and illicit (in red) data points without *keep-alive* messages that was produced with standard behaviors



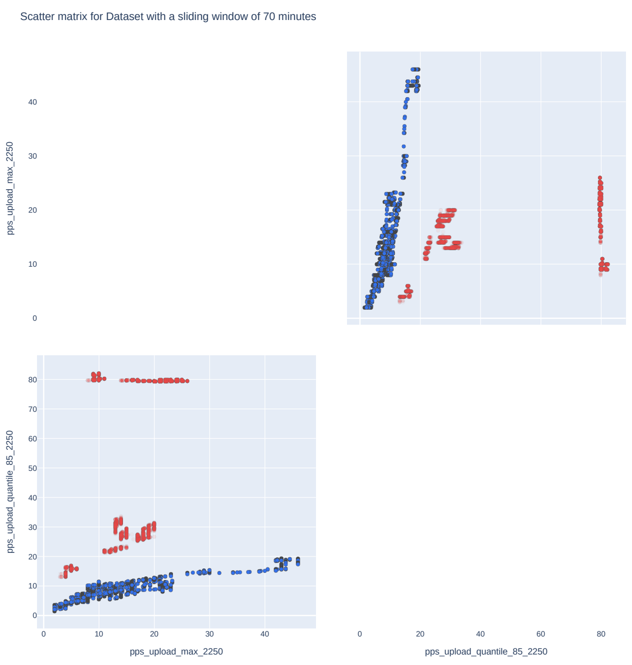
(b) A comparison between licit (in blue) and illicit (in red) data points without *keep-alive messages* that was produced mimicking the licit traffic



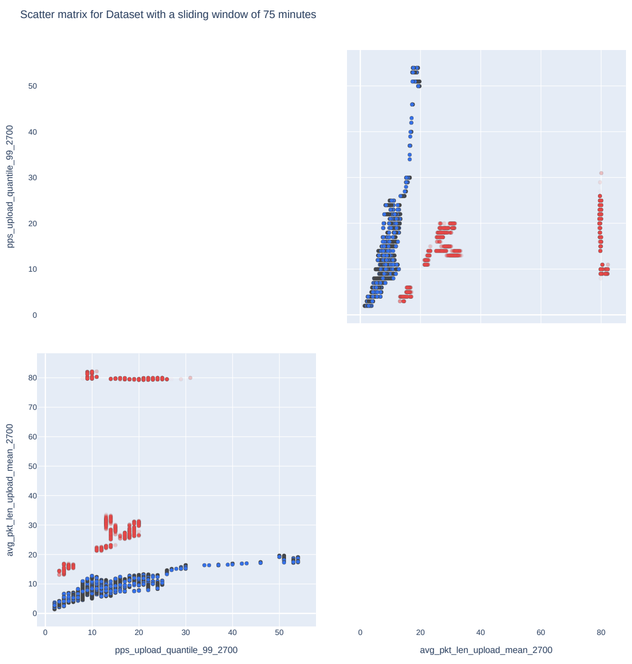
(c) A comparison between licit (in blue) and illicit (in red) data points without *keep-alive messages* that was produced mixed with licit traffic

Figure 4.7: Comparison between features without *keep-alive* messages produced with different behavior approaches

with a sliding window of 70 minutes, which have keep-alive messages, are linearly separable, and the same is applied for features `pps_upload_quantile_99_2700` and `avg_pkt_len_upload_mean_2700`, for the Dataset with a sliding window of 75 minutes.



(a) A comparison between licit (in blue) and illicit (in red) data points with *keep-alive messages* for the Dataset with a sliding window of 70 minutes



(b) A comparison between licit (in blue) and illicit (in red) data points with *keep-alive messages* for the Dataset with a sliding window of 75 minutes

Figure 4.8: Comparison between features with *keep-alive* messages for data points which are linearly separable

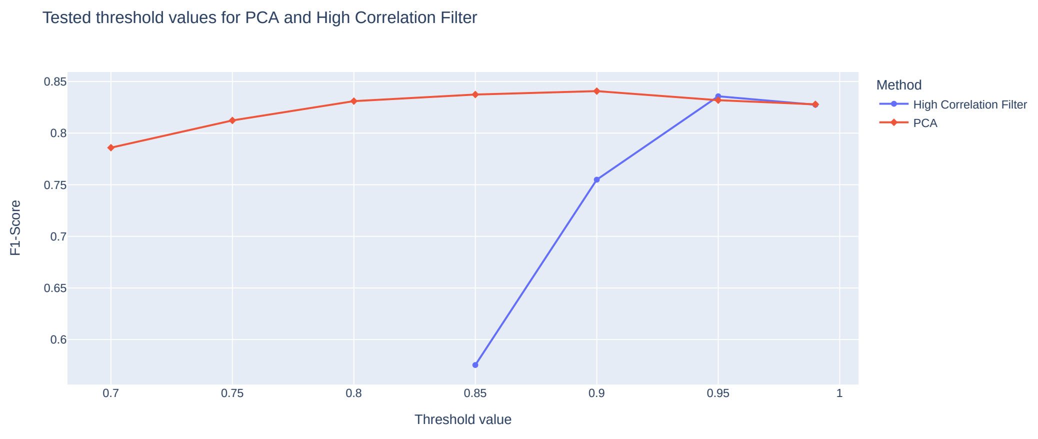
4.4 Dimensionality Reduction

Given the number of features, it is important to consider feature dimensionality techniques that may increase the performance of AD tasks, or speed up the model convergence. As mentioned in section 3.1.4, classic Feature Selection techniques are not indicated for AD as they may overfit the choice of features to our particular set of collected anomalies, where new and unseen bot malware traffic patterns may be more detectable using other set of features that was previously removed. The appropriate way of reducing the dimensionality of our dataset is to use unsupervised dimensionality reduction methods, such as PCA and the High Correlation Filter. We used the Scikit-Learn Python [15] implementation of PCA and our implementation of the High Correlation Filter, as showcased in Figure 3.7, to conduct this process. To determine what percentage of variance to keep, in the case of PCA, and the correlation threshold in the case of High Correlation Filter, one must test multiple values, to later use them to train multiple AD classifiers.

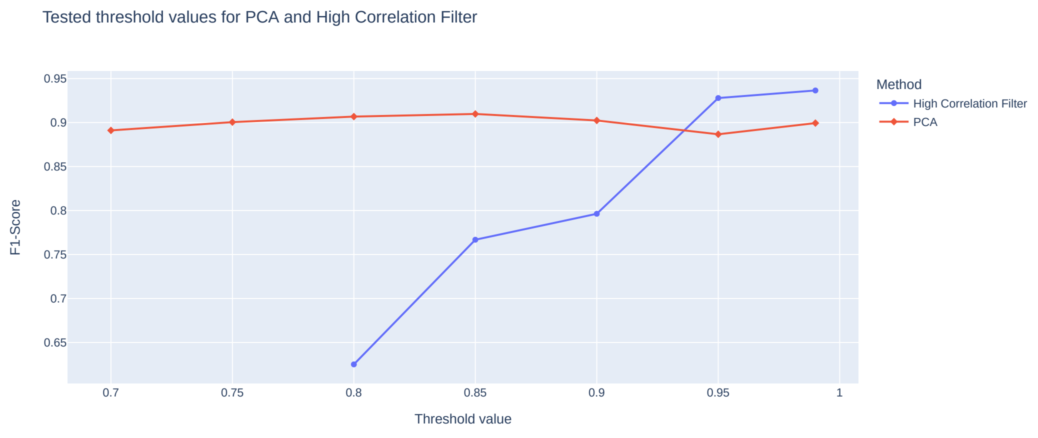
Since AD models need anomalies to test their model of normality, all datasets regarding the different scenarios of bot malware traffic were merged into a single dataset. To test the efficiency of the chosen values, we used the One-Class SVM classifier with the values $k \in 0.85, 0.90, 0.95, 0.98, 0.99$ for the variance percentage and correlation percentage threshold, with 5-fold Cross-Validation. The results for the dataset using a sliding window of 15 minutes depicted in Figure 4.9a show that using a high percentage of variance for PCA, or a high threshold percentage for the High Correlation Filter improves the classification task, however, F1-Score reached its peak when the threshold value was 0.90 for PCA and 0.95 for the High Correlation Filter. For datasets with a sliding value of 70 minutes, Figure 4.9b shows that the best threshold value for PCA is 0.85, and the best threshold value for the High Correlation Filter is 0.99, where for this sliding window it seems to be a better dimensionality reduction technique to be applied, rather than PCA. Finally, for the dataset with a sliding window of 75 minutes, the threshold value of 0.99 is the value that shows the best F1-Score for the PCA and High Correlation Filter approaches, even if it means higher training times.

Because the High Correlation Filter only surpasses PCA when applied in datasets with a sliding window of 70 minutes, and it needs a threshold value closer to 1 to do it so, we decided to test the used AD models without any dimensionality reduction, and to test the same models only with the use of PCA with the a threshold value of 0.85 for datasets with a sliding window of 70 minutes, 0.90 for datasets with a sliding window of 15 minutes, and 0.99 for datasets with a sliding window of 75 minutes.

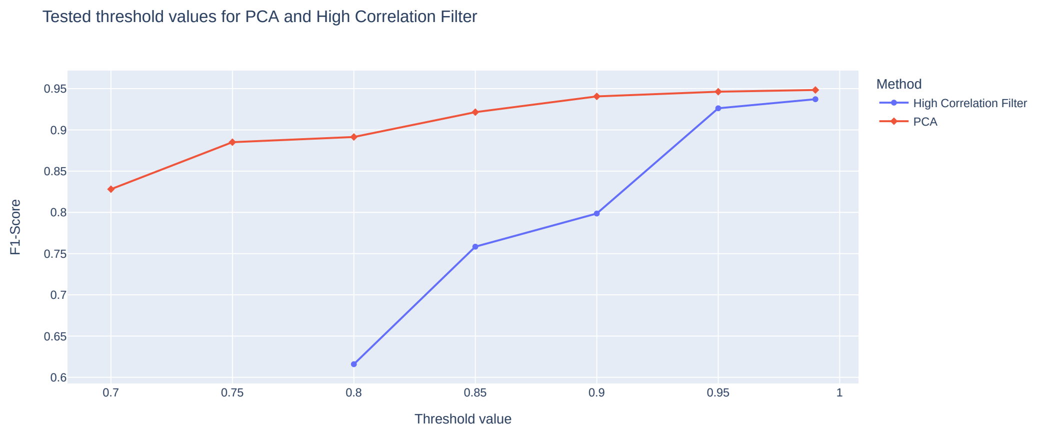
[15] https://scikit-learn.org/stable/



(a) F1-Score for multiple threshold values for the dataset with a sliding window of 15 minutes



(b) F1-Score for multiple threshold values for the dataset with a sliding window of 70 minutes



(c) F1-Score for multiple threshold values for the dataset with a sliding window of 75 minutes

Figure 4.9: F1-Score for multiple threshold values using PCA and the High Correlation Filter

4.5 Classification Results

The final step is to train each AD model with all features, and the selected features with the High Correlation Filter and the selected PCs with PCA, and to record the results, using datasets with different sliding window sizes and different sub-windows. Beforehand, we used the standardization as feature scaling for our data. The tested models were Gaussian Mixture Models (GMM), KDE, LOF, and One-Class SVM, using the Scikit-Learn Python implementations, running on a Jupyter Notebook [16]. For the sliding window size of size 70 minutes, we tested the above models over 4 datasets, namely, a dataset with anomalies from the Scenario 1 and Scenario 2 where illicit traffic use standard behaviors, a dataset with anomalies from the Scenario 3 and Scenario 4 where illicit traffic mimics licit behaviors, a dataset with anomalies from the Scenario 5 and Scenario 6 where low throughput C&C and data exfiltration traffic is mixed with licit traffic, and finally a dataset containing all the above anomalies. *Keep-alive* messages are only present in the dataset that use standard behaviors.

For each dataset and each model, we took the data points from a from a random illicit capture and merged them with an equal amount of licit samples drew from the training data. This set was used to test the best model found in k-fold Cross-Validation in new and unseen bot malware traffic, simulating, therefore, a real-world scenario. The rest of the licit and illicit data points were used as training sets for the k-fold Cross-Validation, using 10 folds ($k = 10$), although illicit instances were only used to create the validation sets for hyper-parameter optimization, and the test sets to evaluate the best model in each fold, both with also an equal amount of licit samples.

Table 4.1 showcases the ranges used for hyper-parameter optimization of thr used AD models.

Model	**Parameter Ranges**
GMM	`n_components`: $[5, 50]$
KDE	`kernel`: gaussian, `bandwidth`: $[0.001, 10]$
LOF	`n_neighbors`: $[2, 50]$
ν-SVM	`kernel`: rbf, `nu`: $[0.01, 0.5]$, `gamma`: $[0.01, 0.5]$

Table 4.1: k-fold

In what concerns training and testing times, we register the time that a model took in hyper-parameter searching, as well as in the classification of test samples, in each fold. Because absolute times depend, for instance, on the hardware, or on the programming language used, we compared the relative times (speedup) of the models to the training and testing times of the GMM model ran without any dimensionality

[16]https://jupyter.org/

reduction technique, and using the dataset with a sliding window of 15 minutes with keep-alive messages. Its average training time was $31m53s \pm 2m37s$ and its average testing time was $4m14s \pm 43s$, with a 95% confidence interval, and all models were run in a machine with an Intel Core i7-1165G7 Processor and 16GB of RAM. The speedup formula used to compute the relative times is given by:

$$S = \frac{T_{ref}}{T_{model}} \tag{4.2}$$

where, T_{ref} is the training, or testing, time of the above GMM model used as reference, and T_{model}, the training, or testing, time of the model being tested.

4.5.1 Dataset with a sliding window of 70 minutes

4.5.1.1 Datasets with keep-alive messages

Keep-alive may create more traffic than the necessary in some scenarios. As we showcased in section 4.3, in this scenario, only two features are enough to able to linearly separate and classify the given data when using "big enough" observation windows. Consequently, we used the features `pps_upload_max_2250` and `pps_upload_quantile_85_2250` from the Dataset with a sliding window of 70 minutes to train the AD classifiers and test them against malware bot traffic that use *keep-alive* messages. One must be aware that using these two features will only work with these particular anomalies with *keep-alive* messages. Other **keep-alive** message patterns might have less linearly separable features, or other linear separable features. Table 4.2 shows that most of the tested AD had no difficulties in correctly classifying the test data, with KDE having an ideal classification score on 10-fold Cross-Validation test sets and GMM and One-Class SVM with very close ideal scores. In term of false positives, KDE and One-Class SVM had no false positives on 10-fold Cross-Validation test sets, with One-Class SVM having the best training time.

		WITH Dimensionality Reduction			
Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	99.07 ± 0.82	$\mathbf{100.0 \pm 0.0}$	55.21 ± 0.76	99.55 ± 0.08
	FPR	1.77 ± 1.56	$\mathbf{0.0 \pm 0.0}$	43.33 ± 0.31	$\mathbf{0.0 \pm 0.0}$
	Training speedup	45.89	61.76	60.35	**689.93**
	Testing speedup	100.0	898.61	**8090.49**	3586.33

Table 4.2: Average results on 10-fold test sets with Dimensionality Reduction for datasets with *keep-alive* messages and with a sliding window of 70 minutes (in percentage, with a 95% confidence interval)

Interestingly, on the test set with new and unseen bot malware traffic, LOF registed a perfect classification score, even if on the 10-fold Cross-Validation its score were far for satisfactory. The rest of the AD also performed very well with no false positives whatsoever.

Type of anomaly		**WITH Dimensionality Reduction**			
		GMM	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	**100.0**	**100.0**	**100.0**	98.77
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.3: Results of tests on new and unseen bot malware traffic with Dimensionality Reduction for datasets with *keep-alive* messages and with a sliding window of 70 minutes

4.5.1.2 Datasets without keep-alive messages

Results shown in Table 4.4 state that most of the tested AD models perform well on the majority types of anomalies, which do not have *keep-alive* messages. For illicit traffic that uses standard behaviors, or that mimicks licit traffic, KDE is the model that shows the best F1-Score and a very low FPR, nonetheless One-Class SVM, or One-Class SVM, also shows an ideal FPR. In addiction, One-Class SVM has the best testing speedup for anomalies with standard behaviors and for anomalies that mimic licit traffic.

In anomalies mixed with licit traffic and when all types of anomalies are present, however, AD models had a decrease in performance, with worst F1-Scores and high FPRs. Nonetheless, KDE shows the best F1-Score and FPR in anomalies mixed with licit traffic, and when all anomalies are present, LOF shows the best F1-Score and One-Class SVM the best FPR. One common aspect to the majority of the results is that models with lower F1-Scores often show higher FPRs, which means that they often produce false alarms, even though they are able to identify most of the anomalies (high Recall).

In terms of training and testing times, One-Class SVM has the best testing times for anomalies with standard behaviors and anomalies that mimic licit traffic, however LOF has the best training and test times for anomalies mixed with licit traffic and when all anomalies are present.

The results of tests on new and unseen bot malware traffic (Table 4.5) show that, once again, KDE was the model with the best F1-Scores and FPRs for anomalies with stantard behaviors and anomalies that mimic licit traffic. When the anomalies were mixed with licit traffic, boundary-based AD models, namely LOF and One-Class SVM, had the worst results, with low F1-Scores and a high number of false positives. For this type of anomalies, GMM was the model with the best F1-Score and the only without any false alarms. When all anomalies are present, GMM was also the model with the best F1-Score, but all models were able to correctly classify every licit data point.

When PCA is active (Table 4.6), keeping 85% of variance, some models slightly increased their performance, while others slightly decreased, although in most of them,

		WITHOUT Dimensionality Reduction			
Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	97.83 ± 1.27	$\mathbf{99.99 \pm 0.01}$	99.75 ± 0.09	98.86 ± 0.21
	FPR	3.95 ± 2.29	$\mathbf{0.0 \pm 0.0}$	0.10 ± 0.08	$\mathbf{0.0 \pm 0.0}$
	Training speedup	4.34	15.58	**15.72**	4.93
	Testing speedup	4.73	23.30	33.47	**234.07**
mimicking licit traffic	*F1-Score*	99.08 ± 0.71	$\mathbf{99.93 \pm 0.05}$	98.95 ± 0.23	97.14 ± 0.32
	FPR	1.29 ± 1.28	$\mathbf{0.06 \pm 0.07}$	0.82 ± 0.41	3.46 ± 0.44
	Training speedup	4.66	**85.32**	2.18	5.36
	Testing speedup	9.60	22.71	35.55	**269.18**
mixed with licit traffic	*F1-Score*	88.93 ± 1.56	$\mathbf{92.28 \pm 0.41}$	81.91 ± 0.72	84.26 ± 0.73
	FPR	14.41 ± 2.88	$\mathbf{4.97 \pm 0.80}$	17.36 ± 1.78	12.93 ± 0.88
	Training speedup	4.44	16.49	**16.92**	5.85
	Testing speedup	11.84	38.98	**41.86**	33.67
all anomalies	*F1-Score*	75.10 ± 1.49	88.03 ± 0.85	$\mathbf{89.44 \pm 0.55}$	87.69 ± 0.52
	FPR	31.89 ± 1.17	18.04 ± 1.36	11.77 ± 1.52	$\mathbf{6.58 \pm 2.38}$
	Training speedup	3.90	17.11	**18.08**	6.60
	Testing speedup	4.39	24.56	**42.86**	41.48

Table 4.4: Average results on 10-fold test sets without Dimensionality Reduction for datasets without *keep-alive* messages and with a sliding window of 70 minutes (in percentage, with a 95% confidence interval)

Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	99.95	**100.0**	99.20	98.97
	FPR	**0.0**	**0.0**	1.28	**0.0**
mimicking licit traffic	*F1-Score*	99.84	**99.98**	99.19	89.04
	FPR	**0.0**	**0.0**	**0.0**	16.82
mixed with licit traffic	*F1-Score*	**95.96**	89.48	78.87	79.52
	FPR	**0.0**	8.69	22.03	20.30
all anomalies	*F1-Score*	**99.85**	98.07	94.28	89.29
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.5: Results of tests on new and unseen bot malware traffic without Dimensionality Reduction for datasets without *keep-alive* messages and with a sliding window of 70 minutes

their training and testing speedups vastly increased, meaning that using PCA is more computationally efficient. In anomalies that use standard behaviors, that mimic licit traffic, and that are mixed with licit traffic, KDE continues to be the model that has the best F1-Score, and obtained the best testing time in anomalies that mimic licit traffic, however, its FPR slightly increased. The FPR of One-Class SVM remains to be ideal for anomalies with standard behaviors, and LOF was the model that had the best testing time. When we tested the chosen AD models with all anomalies, One-Class SVM obtained the best F1-Score, although, that F1-Score is lower when comparing with the best F1-Scores when the different types of anomalies where tested separately.

Its FPR was also a bit high, although lesser than the best FPR when bot malware traffic is mixed with licit traffic. LOF obtained the best testing times in last type of anomalies and when all anomalies are present.

		WITH PCA			
Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	99.42 ± 0.46	$\mathbf{99.77 \pm 0.17}$	98.41 ± 0.23	98.26 ± 0.16
	FPR	0.94 ± 0.90	0.36 ± 0.33	1.50 ± 0.3	$\mathbf{0.0 \pm 0.0}$
	Training speedup	34.96	54.36	56.55	**62.72**
	Testing speedup	90.51	654.12	**1559.69**	1058.02
mimicking licit traffic	*F1-Score*	98.07 ± 0.31	$\mathbf{99.43 \pm 0.09}$	96.71 ± 0.39	95.24 ± 0.38
	FPR	2.24 ± 0.60	$\mathbf{0.35 \pm 0.15}$	2.27 ± 0.66	3.30 ± 0.31
	Training speedup	30.72	49.71	54.95	**57.86**
	Testing speedup	59.98	**6665.90**	2482.30	550.40
mixed with licit traffic	*F1-Score*	70.67 ± 0.74	$\mathbf{92.25 \pm 0.54}$	82.58 ± 0.73	85.66 ± 0.58
	FPR	29.28 ± 1.77	$\mathbf{5.86 \pm 0.94}$	17.75 ± 2.11	12.26 ± 0.59
	Training speedup	32.45	48.36	46.20	**52.99**
	Testing speedup	58.76	606.20	**3426.20**	142.22
all anomalies	*F1-Score*	86.72 ± 0.81	82.16 ± 0.94	88.53 ± 0.48	$\mathbf{88.65 \pm 0.47}$
	FPR	18.57 ± 1.39	25.49 ± 1.02	11.71 ± 1.41	$\mathbf{4.19 \pm 0.60}$
	Training speedup	36.89	60.02	57.69	**68.67**
	Testing speedup	63.13	508.22	**3594.42**	171.01

Table 4.6: Average results on 10-fold test sets with PCA for datasets without *keep-alive* messages and with a sliding window of 70 minutes (in percentage, with a 95% confidence interval)

When the above methods were tested against new and unseen bot malware traffic (Table 4.7), results where in pair with the Cross-Validation scores, with FPRs being ideal most of the time, however, in anomalies mixed with licit traffic, FPRs where higher than those reported in Cross-Validation. KDE was the model that performed best in all types of anomalies.

Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	99.68	**99.76**	99.26	97.98
	FPR	**0.0**	**0.0**	**0.0**	**0.0**
mimicking licit traffic	*F1-Score*	99.17	**99.62**	98.24	96.69
	FPR	**0.0**	**0.0**	**0.0**	**0.0**
mixed with licit traffic	*F1-Score*	73.68	**90.35**	71.09	82.87
	FPR	27.50	**10.01**	32.36	16.61
all anomalies	*F1-Score*	97.39	**99.02**	93.49	89.49
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.7: Results of tests on new and unseen bot malware traffic with PCA for datasets without *keep-alive* messages and with a sliding window of 70 minutes

4.5.2 Dataset with a sliding window of 15 minutes

4.5.2.1 Datasets with keep-alive messages

As showcased in section Dataset Exploration, since for observation windows with smaller sizes and malware traffic with *keep-alive* messages we do not have data with enough deviations to be linearly separable, we decided to train and test the models without any dimensionality reduction by keeping all features and with PCA keeping 90% of the variance.

The results displayed in tables 4.8 and 4.9 show that LOF obtained the best F1-Score on 10-fold test sets with very FPRs, however, its F1-Score on the test with new and unseen bot malware traffic was the lowest among all tested AD models. One-Class SVM did not report any false positives whatsoever and had a great F1-Score in both Cross-Validation and new and unseen bot malware test set, being also the model with the fastest testing times.

		WITHOUT Dimensionality Reduction			
Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	76.42 ± 0.35	76.61 ± 0.17	$\mathbf{99.41 \pm 0.04}$	99.32 ± 0.07
	FPR	30.93 ± 0.29	30.77 ± 0.14	0.06 ± 0.13	$\mathbf{0.0 \pm 0.0}$
	Training speedup	1.0	**4.47**	3.43	0.81
	Testing speedup	1.0	3.79	6.86	**63.5**

Table 4.8: Average results on 10-fold test sets without Dimensionality Reduction for datasets with *keep-alive* messages and with a sliding window of 15 minutes (in percentage, with a 95% confidence interval)

Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	**100.0**	**100.0**	83.20	97.51
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.9: Results of tests on new and unseen bot malware traffic without Dimensionality Reduction for datasets with *keep-alive* messages and with a sliding window of 15 minutes

When training and testing the models with PCA, results in tables 4.10 and 4.11 show that was the One-Class SVM, this time, that obtained the best F1-Score on 10-fold Cross-Validation, with no false positives, and, again, with the best testing time. On the test set with new and unseen bot malware traffic, GMM and KDE obtained ideal F1-Scores and no AD model reported any false positives whatsoever.

Type of anomaly		WITH PCA GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	76.39 ± 0.45	76.31 ± 0.50	89.21 ± 0.57	**99.35 ± 0.09**
	FPR	30.96 ± 0.37	31.02 ± 0.41	17.17 ± 0.71	**0.0 ± 0.0**
	Training speedup	18.94	**38.26**	35.43	18.22
	Testing speedup	25.4	84.67	875.86	**976.92**

Table 4.10: Average results on 10-fold test sets with PCA for datasets with *keep-alive* messages and with a sliding window of 15 minutes (in percentage, with a 95% confidence interval)

Type of anomaly		GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	**100.0**	**100.0**	99.53	99.26
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.11: Results of tests on new and unseen bot malware traffic with PCA for datasets with *keep-alive* messages and with a sliding window of 15 minutes

4.5.2.2 Datasets without keep-alive messages

With a sliding window of 15 minutes, anomalies with standard behaviors and that mimic licit traffic reported great performance on detecting these type of anomalies (Table 4.12), nonetheless, for anomalies that mimic licit traffic, FPRs slightly increased. For anomalies mixed with licit traffic and when all anomalies were present, performance results were suboptimal with KDE having the highest detection rate, but lower FPRs than the LOF and One-Class SVM algorithms.

Type of anomaly		WITHOUT Dimensionality Reduction GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	99.57 ± 0.21	**99.63 ± 0.10**	98.95 ± 0.16	95.58 ± 0.42
	FPR	0.35 ± 0.41	0.43 ± 0.26	0.65 ± 0.31	**0.0 ± 0.0**
	Training speedup	1.1	3.13	**10.99**	2.99
	Testing speedup	1.38	5.91	23.09	**63.5**
mimicking licit traffic	*F1-Score*	93.12 ± 0.98	**97.73 ± 0.18**	94.86 ± 0.22	90.05 ± 0.37
	FPR	7.15 ± 2.71	**1.54 ± 0.33**	3.76 ± 0.61	1.77 ± 0.28
	Training speedup	2.64	10.28	**10.69**	3.32
	Testing speedup	3.26	13.37	**23.09**	10.16
mixed with licit traffic	*F1-Score*	70.27 ± 1.34	**81.01 ± 0.32**	71.35 ± 0.58	73.02 ± 0.46
	FPR	32.71 ± 1.97	15.71 ± 1.99	31.73 ± 1.10	**10.92 ± 0.25**
	Training speedup	2.91	8.14	**10.75**	3.07
	Testing speedup	3.63	15.88	**25.4**	2.25
all anomalies	*F1-Score*	84.88 ± 1.20	**84.89 ± 0.59**	84.47 ± 0.55	81.72 ± 0.34
	FPR	19.31 ± 1.68	21.32 ± 1.23	**18.73 ± 1.68**	19.11 ± 1.23
	Training speedup	3.57	11.59	**13.1**	3.63
	Testing speedup	5.29	18.14	**28.2**	14.11

Table 4.12: Average results on 10-fold test sets without Dimensionality Reduction for datasets without *keep-alive* messages and with a sliding window of 15 minutes (in percentage, with a 95% confidence interval)

Testing the best models obtained in Cross-Validation with data containing new and unseen malware traffic (Figure 4.13), results show that for anomalies with standard behaviors and anomalies that mimic licit traffic almost all models reported ideal FPRs with great detection rates, however for anomalies mixed with licit traffic and when all anomalies were present, the detection rates and the number of false positives were suboptimal.

Type of anomaly		**GMM**	**KDE**	**LOF**	**One-Class SVM**
standard behaviors	*F1-Score*	**99.99**	98.20	99.29	95.41
	FPR	**0.0**	**0.0**	0.04	**0.0**
mimicking licit traffic	*F1-Score*	94.99	**98.46**	95.60	90.94
	FPR	**0.0**	**0.0**	**0.0**	**0.0**
mixed with licit traffic	*F1-Score*	41.11	**73.64**	70.39	69.15
	FPR	49.94	28.49	33.32	**18.74**
all anomalies	*F1-Score*	64.22	40.23	**77.84**	40.62
	FPR	39.07	48.87	**26.84**	51.65

Table 4.13: Results of tests on new and unseen bot malware traffic without Dimensionality Reduction for datasets without *keep-alive* messages and with a sliding window of 15 minutes

When using PCA as method for dimensionality reduction in datasets with a sliding window of 15 minutes, Table 4.14 shows that the detection rates and FPRs were in general slightly worse than using all features, however for anomalies that mimic licit traffic, the best performing model, KDE, increased its performance and its training and testing times. tfor

Testing the models that performed best with data with new and unseen malware bot traffic, in anomalies with standard behaviors, KDE was the best performing model, however all models reported ideal FPRs, and in anomalies that mimic licit traffic, GMM obtained the lowest FPR. KDE was the model with the best detection rated for all kinds of anomalies, and One-Class SVM obtained an ideal FPR also when all anomalies were present.

In sum, comparing to the datasets with a sliding window of 70 minutes, using datasets with smaller observation windows decreased the performance of Novelty models, nonetheless, it is still possible to have great detection rates with low false positives with anomalies that use standard behaviors and anomalies that mimic licit traffic, being faster to detect those type of anomalies. In anomalies mixed with licit traffic, however, training these Novelty models with these window sizes did not produce optimal results.

Type of anomaly		WITH PCA GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	**98.38 ± 0.30**	97.46 ± 1.71	96.35 ± 0.41	97.21 ± 0.21
	FPR	**1.38 ± 0.88**	3.87 ± 3.22	2.72±	1.62 ± 0.25
	Training speedup	19.52	32.98	**34.97**	19.32
	Testing speedup	25.4	254.00	301.20	**409.68**
mimicking licit traffic	*F1-Score*	85.38 ± 0.48	**98.03 ± 0.18**	92.57 ± 0.27	86.44 ± 0.63
	FPR	16.94 ± 1.20	**1.33 ± 0.22**	5.61 ± 0.40	13.12 ± 4.32
	Training speedup	20.79	37.14	**39.04**	21.26
	Testing speedup	28.22	**3629**	983.75	73.09
mixed with licit traffic	*F1-Score*	54.49 ± 0.87	**78.99 ± 0.55**	68.73 ± 1.49	71.82 ± 0.56
	FPR	45.10 ± 0.95	**19.24 ± 1.42**	33.91 ± 1.82	28.37 ± 0.70
	Training speedup	19.32	28.55	**37.51**	19.78
	Testing speedup	33.55	254	**1788.73**	32.31
all anomalies	*F1-Score*	78.17 ± 0.36	81.07 ± 2.09	**83.72 ± 0.52**	81.67 ± 0.35
	FPR	27.80 ± 0.54	26.30 ± 2.09	19.47 ± 0.99	**11.01 ± 0.41**
	Training speedup	20.35	35.15	**36.99**	17.11
	Testing speedup	28.22	371.39	**1137.64**	15.34

Table 4.14: Average results on 10-fold test sets with PCA for datasets without *keep-alive* messages and with a sliding window of 15 minutes (in percentage, with a 95% confidence interval)

Type of anomaly		GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	98.94	**99.41**	97.82	97.93
	FPR	**0.0**	**0.0**	**0.0**	**0.0**
mimicking licit traffic	*F1-Score*	93.87	**98.09**	94.47	95.12
	FPR	**0.60**	0.73	0.81	1.13
mixed with licit traffic	*F1-Score*	43.95	**82.05**	77.85	73.88
	FPR	54.12	**10.71**	22.68	24.73
all anomalies	*F1-Score*	95.92	**98.93**	93.06	85.46
	FPR	0.52	0.17	3.86	**0.0**

Table 4.15: Results of tests on new and unseen bot malware traffic with PCA for datasets without *keep-alive* messages and with a sliding window of 15 minutes

4.5.3 Datasets with a sliding window of 75 minutes

4.5.3.1 Datasets with keep-alive messages

Similar to section 4.5.1.1, using bigger observation windows make the illicit data more divergent from the licit instances, and so, as showcased in section 4.3, we only need two features for data to be linearly separable. For datasets using a sliding window of 75 minutes, the features chosen to train the AD algorithms with linearly separable data were `pps_upload_quantile_99_2700` and `avg_pkt_len_upload_mean_2700`. Table 4.16 shows that the best performing model was the KDE with ideal detection and false positive rates, however, the One-Class SVM obtained the best training and testing times with no false positives on Cross-Validation.

Type of anomaly		GMM	KDE	LOF	One-Class SVM
		WITH Dimensionality Reduction			
standard behaviors	*F1-Score*	99.67 ± 0.29	$\mathbf{100.0 \pm 0.0}$	99.78 ± 0.10	99.44 ± 0.03
	FPR	0.64 ± 0.57	$\mathbf{0.0 \pm 0.0}$	0.10 ± 0.11	$\mathbf{0.0 \pm 0.0}$
	Training speedup	52.26	59.72	58.83	**94.65**
	Testing speedup	144.88	834.15	2495.39	**3898.37**

Table 4.16: Average results on 10-fold test sets with Dimensionality Reduction for datasets with *keep-alive* messages and with a sliding window of 75 minutes (in percentage, with a 95% confidence interval)

In the test set with new and unseen malware bot traffic with *keep-alive* messages, all models performed as expected with KDE and GMM having ideal detection rates and FPRs.

Type of anomaly		GMM	KDE	LOF	One-Class SVM
		WITH Dimensionality Reduction			
standard behaviors	*F1-Score*	**100.0**	**100.0**	99.95	99.44
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.17: Results of tests on new and unseen bot malware traffic with Dimensionality Reduction for datasets with *keep-alive* messages and with a sliding window of 75 minutes

4.5.3.2 Datasets without keep-alive messages

With a sliding window of 75 minutes and without *keep-alive* messages in malware bot traffic (Table 4.18), KDE was the model that had the best detection rates in all types of anomalies, however, when all anomalies were present LOF obtained the best detection rate. In anomalies that use standard behaviors KDE and One-Class SVM obtained ideal FPRs, with One-Class SVM obtaining again an ideal FPR in anomalies that mimic licit traffic and having the best testing times in all situations.

In Table 4.19, testing the best model instances with new and unseen bot malware traffic reveals that in anomalies with standard behaviors, GMM matched the detection and false positive rates of KDE, and in general, KDE obtained the best detection rates. However, when all anomalies are present, GMM obtained a detection rate greater than KDE.

When PCA was used (Table 4.20), KDE continued to be the model that had the best detection rates in the three types of anomalies, having slightly better detection rates with PCA in anomalies using standard behaviors and that mimic licit traffic than without dimensionality reduction, however, in anomalies mixed with licit traffic, the

Type of anomaly		WITHOUT Dimensionality Reduction GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	98.83 ± 0.97	**100.0 ± 0.0**	$99.85 \pm .05$	99.27 ± 0.30
	FPR	2.16 ± 1.81	**0.0 ± 0.0**	0.22 ± 0.09	**0.0 ± 0.0**
	Training speedup	10.27	**24.76**	22.45	18.31
	Testing speedup	11.34	43.89	54.58	**638.44**
mimicking licit traffic	*F1-Score*	99.73 ± 0.23	**99.90 ± 0.04**	99.47 ± 0.13	98.70 ± 0.20
	FPR	0.46 ± 0.47	0.09 ± 0.08	0.38 ± 0.26	**0.0 ± 0.0**
	Training speedup	9.25	19.45	**20.86**	18.66
	Testing speedup	21.93	34.42	49.24	**412.43**
mixed with licit traffic	*F1-Score*	84.43 ± 3.05	**98.02 ± 0.16**	90.93 ± 0.59	93.13 ± 0.31
	FPR	21.90 ± 3.31	**0.50 ± 0.24**	7.16 ± 1.29	3.75 ± 0.26
	Training speedup	5.62	8.66	**10.84**	7.29
	Testing speedup	8.01	17.35	22.31	**44.94**
all anomalies	*F1-Score*	76.88 ± 2.14	86.16 ± 0.58	**93.50 ± 0.40**	92.57 ± 0.34
	FPR	30.39 ± 1.79	20.93 ± 0.76	6.64 ± 1.02	**1.10 ± 0.16**
	Training speedup	11.43	**22.69**	19.89	14.82
	Testing speedup	12.33	36.49	44.92	**64.67**

Table 4.18: Average results on 10-fold test sets without Dimensionality Reduction for datasets without *keep-alive* messages and with a sliding window of 75 minutes (in percentage, with a 95% confidence interval)

Type of anomaly		GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	**100.0**	**100.0**	99.06	99.32
	FPR	**0.0**	**0.0**	1.81	**0.0**
mimicking licit traffic	*F1-Score*	99.97	**99.98**	99.59	98.44
	FPR	**0.0**	**0.0**	**0.0**	**0.0**
mixed with licit traffic	*F1-Score*	96.97	**98.54**	94.51	94.64
	FPR	5.02	**0.0**	**0.0**	**0.0**
all anomalies	*F1-Score*	**99.98**	98.92	96.00	92.99
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.19: Results of tests on new and unseen bot malware traffic with PCA for datasets without *keep-alive* messages and with a sliding window of 70 minutes

model registed a FPR greater than 1%. One-Class SVM also increased its detection rates being the model with the highest detection rate and the lowest FPR when all anomalies are present, and it was the model had, overall, the best training and testing times.

Testing the best performing models in Cross-Validation with new and unssen bot malware traffic (Table 4.21) shows than, once again, KDE the model that had the best detection and false positive rates overall. In anomalies with standard behaviors, GMM had also an ideal detection and false positive rate, and when all anomalies were present, GMM was the best performing model.

In sum, comparing to the datasets with a sliding window of 70 minutes and the

Type of anomaly		GMM	KDE	LOF	One-Class SVM
		WITH PCA			
standard behaviors	*F1-Score*	99.73 ± 0.45	$\mathbf{100.0 \pm 0.0}$	99.89 ± 0.07	99.36 ± 0.13
	FPR	0.50 ± 0.85	$\mathbf{0.0 \pm 0.0}$	0.06 ± 0.08	$\mathbf{0.0 \pm 0.0}$
	Training speedup	31.58	48.03	27.15	**59.95**
	Testing speedup	47.43	194.71	76.67	**1967.65**
mimicking licit traffic	*F1-Score*	97.83 ± 2.04	$\mathbf{99.93 \pm 0.04}$	99.24 ± 0.21	98.86 ± 0.17
	FPR	3.72 ± 3.52	0.11 ± 0.08	0.46 ± 0.18	$\mathbf{0.0 \pm 0.0}$
	Training speedup	32.06	52.04	24.99	**59.01**
	Testing speedup	66.59	1255.81	68.92	**1342.73**
mixed with licit traffic	*F1-Score*	92.76 ± 1.82	$\mathbf{97.46 \pm 0.29}$	89.12 ± 0.60	91.86 ± 0.38
	FPR	9.62 ± 3.29	$\mathbf{1.55 \pm 0.36}$	9.71 ± 1.18	7.08 ± 0.47
	Training speedup	29.97	45.10	26.00	**54.67**
	Testing speedup	52.96	**544.60**	68.41	318.70
all anomalies	*F1-Score*	76.84 ± 0.65	85.22 ± 0.64	93.15 ± 0.38	$\mathbf{93.63 \pm 0.47}$
	FPR	30.57 ± 0.55	22.13 ± 0.78	6.57 ± 0.94	$\mathbf{2.51 \pm 0.36}$
	Training speedup	28.33	48.11	26.61	**60.39**
	Testing speedup	70.26	194.96	71.14	**326.28**

Table 4.20: Average results on 10-fold test sets with PCA for datasets without *keep-alive* messages and with a sliding window of 75 minutes (in percentage, with a 95% confidence interval)

Type of anomaly		GMM	KDE	LOF	One-Class SVM
standard behaviors	*F1-Score*	**100.0**	**100.0**	88.57	99.46
	FPR	**0.0**	**0.0**	18.38	**0.0**
mimicking licit traffic	*F1-Score*	99.95	**99.98**	99.64	98.73
	FPR	**0.0**	**0.0**	**0.0**	**0.0**
mixed with licit traffic	*F1-Score*	87.14	**96.34**	85.36	90.12
	FPR	18.48	**4.12**	17.10	9.56
all anomalies	*F1-Score*	**99.98**	99.54	95.99	94.42
	FPR	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.21: Results of tests on new and unseen bot malware traffic with PCA for datasets without *keep-alive* messages and with a sliding window of 75 minutes

datasets with a sliding window of 15 minutes, using larger observation windows increased the performance of Novelty models, achieving higher detection rates and very low false positive rates, even for anomalies that generate very low throughput rates.

4.6 Conclusion

Training Machine Learning classifiers on every type of existing bot malware behaviors is realistically not possible. This work covered only a few possible behaviors that malware bots might have with the objective to evade the current security measures, although many other possible illicit behaviors may arise. In order to create a reliable model, even for anomalies that one is not aware of, we trained AD models that resort in Novelty Detection techniques and that only need to be familiar with licit behaviors, modelling a representation of normality. If by any means, legitimate DNS usages change, or if new ones arise, they only have to be re-trained with the new licit data.

The reported results show that in anomalies that have regular throughput rates, namely anomalies that use standard behaviors and anomalies that mimic licit traffic, most AD models perform very well, with satisfactory F1-Scores and very low false alarms, nonetheless, for low throughput bot malware traffic were bots only send a few packets from time to time, it is still possible to have satisfactory results with a FPR $< 1\%$ using KDE. More moments of decision with larger observation windows may be needed to evaluate if the rest of the tested classical Novelty Detection algorithms can generate satisfactory FPRs for these anomalies using low throughput rates.

These tests were conducted in with a licit dataset that was produced by capturing licit behaviors from multiple operating systems, browser applications and configurations. Nonetheless, all these licit network traces only represent different behaviors from one user, and so, it would be advantageous to gather more behaviors from multiple users, where density-based AD algorithms may use them to create denser distributions of normality and boundary-based AD algorithms to create more precise decision boundaries.

CHAPTER 5

Conclusions and future work

The study in chapter 2 showed that the DNS infraestructure is vulnerable by default to illicit usages. Since it is a critical part of the Internet, multiple studies regarding the usage of DNS as a covert channel were conducted, aiming to identify the illicit communications of malware bots and botnets. Notwithstanding, they all rely on DPI techniques, such as analysing DNS queries entropy's, which may contain sensitive information. Moreover, these methodologies are ineffective when encrypted DNS traffic is being used inside a corporate network. On the other hand, the proposed AD mechanisms focus on the behavioral and temporal characteristics of DNS traffic instead of DNS payloads, being more robust to a wider range of possible illicit usages of DNS.

These methodologies extract statistical features from the captured network metrics that might reveal divergent behaviors between licit and illicit traffic patterns and different underlying silence and activity periods. Furthermore, the usage of Novelty Detection algorithms for AD tasks enables us to only model the typical DNS behaviors, so that the trained classifier is not bound to any specific type of malware bot behavior, with the addition of being robust to new and unseen phenomena. This class of Machine Learning algorithms, although, cannot be optimized with standard Feature Selection methods, but instead with unsupervised dimensionality reduction techniques, since Novelty Detection models fundamentally train in an unsupervised way, and they must not have prior knowledge regarding malware behaviors. A common unsupervised dimensionality reduction technique is the PCA, however, we also proposed an alternative unsupervised solution for AD using a High Correlation Filter.

To validate those mechanisms to detect illicit DNS usages, we designed 6 scenarios of possible bot malware behaviors. The first two scenarios were designed to replicate C&C communications with Push mechanisms and data exfiltration using parametric statistical distributions similar to the collected licit ones. Scenarios 5 and 6 have the same purpose, however we used a different method to mimic licit traffic, which is collecting the licit traffic metrics and generating packet rates and packet lengths according to the empirical probabilities processed from those licit captures. This is essentially a non-parametric way of mimicking licit traffic. Finally, the two last scenarios were produced to cover those cases where malware bots generate C&C and data exfiltration traffic by sending fewer packets with greater silence periods in between, aiming to be blended with licit traffic. These type of low throughput illicit communications are very difficult to detect with the current mechanisms.

In order to test those scenarios with the chosen Novelty Detection algorithms, Scenarios 1 and 2 were grouped in *anomalies with standard behaviors*, Scenarios 3 and 4 in *anomalies mimicking licit traffic*, and Scenarios 5 and 6 in *anomalies mixed with licit traffic*. We also tested those models with the presence of all types of anomalies, and with the presence of *keep-alive* messages, for the case of anomalies with standard behaviors, all with different sliding windows with multiple observation sub-windows, aiming to capture different perspectives of the collected data.

In the first type of anomalies, the Novelty Detection models were able to detect the anomalous behaviors with ease when the illicit data did not contained *keep-alive* messages. Some models were able to obtain very high detection rates (98% – 100%) and an ideal FPRs, with and without PCA, with good training and testing times, as it was the case of KDE for a 99% detection rate and One-Class SVM with an ideal 0% FPR. When *keep-alive* messages are present, boundary-based models outperformed density-based models with One-Class SVM reaching 98% detection rates with smaller sliding windows, and with larger sliding windows, it was possible to achieve ideal detection rates with no false positives.

Anomalies that mimic licit behaviors were also effortless to detect with detection rates ranging between 97% and 99% with and without PCA. This time FPRs were a bit higher, meaning that licit behaviors were more mistook with illicit samples, however some FPRs were still under 1%.

Being more difficult to detect, the detection rates for anomalies with low throughput rates that were mixed with licit traffic were satisfactory, with the best detection rate reaching 98% with a FPR lower than 1%, however the FPRs for most classifiers were higher than the other two types of anomalies. Besides that, tests on new and unseen malware bot traffic were very satisfactory, with the best detection rate being 98% with an ideal FPR.

When traffic from all types of anomalies were present in datasets, the detection rates ranged between 83% and 93%, i.e. overall lower than when only one type of anomaly was present. False positives increased, however, it was still possible to achieve FPRs in the order of 1%. The reason for these lower performance metrics is that some anomalies may require different hyper-parameter settings. We conclude that using hyper-parameter searching may not be the best practice for AD, as it may overfit particular anomalies. Using random hyper-parameter settings, or choosing hyper-parameter settings with proven efficacy in this particular problem of AD is more adequate.

In conclusion, these tests showed the viability of the methodologies proposed in chapter 3, even if it is impossible to check all kinds of bot malware behaviors, and sets the ground for the development of more robust and privacy focused solutions for bot malware and botnet detection that rely in AD using network traffic behaviors instead of analyzing DNS query payloads and other DPI techniques.

5.1 Future work

Anomaly Detection is a vast field where we barely scratched the surface. In our work, we tested our licit and illicit data with four classic Novelty Detection algorithms, however, many other algorithms can be tried out, for instance, by using Deep Anomaly Detection models such as Neural Generative Models and Normalizing Flows, which might be useful to detect malware bots that use low throughput rates with higher detection rates, lower FPRs and smaller observation windows.

Popular DNS Tunneling tools are designed to be as efficient as possible, with high upstream and downstream throughputs and low latency so that surfing the web, or downloading files at a reasonable time over DNS, is possible. Although, they often achieve it by generating large DNS packets aiming to reduce network congestion, which can be an inconvenient for a malware bot that attempts to mimic a licit behavior. *Keep-alive* messages are also an inconvenient as they can induce additional overhead to these tools. In this work, we resolved the problem of bigger DNS packet sizes than usual by editing the source code of DNS2TCP, however, it would be interesting to have proper DNS Tunneling tools specifically designed to emulate licit DNS usages with realistic packet sizes and higher configuration capabilities, such as the possibility to configure RRs, encoding schemes, and disable *keep-alive* messages.

References

[1] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the dns protocol," *Computers & Security*, vol. 80, pp. 36–53, 2019, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.09.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818304000.

[2] C. Wilson, "Botnets, cybercrime, and cyberterrorism: Vulnerabilities and policy issues for congress," LIBRARY OF CONGRESS WASHINGTON DC CONGRESSIONAL RESEARCH SERVICE, 2008.

[3] K. Alieyan, A. ALmomani, A. Manasrah, and M. M. Kadhum, "A survey of botnet detection based on dns," *Neural Computing and Applications*, vol. 28, no. 7, pp. 1541–1558, 2017.

[4] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, 2009, pp. 268–273. DOI: 10.1109/SECURWARE.2009.48.

[5] A. Karim, R. B. Salleh, M. Shiraz, S. A. A. Shah, I. Awan, and N. B. Anuar, "Botnet detection techniques: Review, future trends, and issues," *Journal of Zhejiang University SCIENCE C*, vol. 15, no. 11, pp. 943–983, Nov. 2014, ISSN: 1869-196X. DOI: 10.1631/jzus.C1300242. [Online]. Available: https://doi.org/10.1631/jzus.C1300242.

[6] H. K. et al, "Security of cyber-physical systems," in *The Structure of Botnets*, 2020, pp. 88–103. DOI: 10.1007/978-3-030-45541-5.

[7] D. Ramsbrock and X. Wang, "The botnet problem," in Dec. 2013, pp. 223–238, ISBN: 9780123943972. DOI: 10.1016/B978-0-12-394397-2.00012-X.

[8] N. Negash and X. Che, "An overview of modern botnets," *Information Security Journal: A Global Perspective*, vol. 24, no. 4-6, pp. 127–132, 2015. DOI: 10.1080/19393555.2015.1075629. eprint: https://doi.org/10.1080/19393555.2015.1075629. [Online]. Available: https://doi.org/10.1080/19393555.2015.1075629.

[9] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A taxonomy of botnet behavior, detection, and defense," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 898–924, 2014. DOI: 10.1109/SURV.2013.091213.00134.

[10] F. L. Greitzer, J. R. Strozer, S. Cohen, A. P. Moore, D. Mundie, and J. Cowley, "Analysis of unintentional insider threats deriving from social engineering exploits," in *2014 IEEE Security and Privacy Workshops*, 2014, pp. 236–250. DOI: 10.1109/SPW.2014.39.

[11] T. Ranka, "Taxonomy of botnet threats," 2006.

[12] A. K. Sood and S. Zeadally, "A taxonomy of domain-generation algorithms," *IEEE Security Privacy*, vol. 14, no. 4, pp. 46–53, 2016. DOI: 10.1109/MSP.2016.76.

[13] G. Ollmann, "Botnet communication topologies," *Retrieved September*, pp. 1–9, 2009.

[14] G. Vormayr, T. Zseby, and J. Fabini, "Botnet communication patterns," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2768–2796, 2017. DOI: 10.1109/COMST.2017.2749442.

[15] E. V. Ruitenbeek and W. H. Sanders, "Modeling peer-to-peer botnets," in *2008 Fifth International Conference on Quantitative Evaluation of Systems*, 2008, pp. 307–316. DOI: 10.1109/QEST.2008.43.

[16] N. Pantic and M. I. Husain, "Covert botnet command and control using twitter," in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC '15, Los Angeles, CA, USA: Association for Computing Machinery, 2015, pp. 171–180, ISBN: 9781450336826. DOI: 10.1145/2818000.2818047. [Online]. Available: https://doi.org/10.1145/2818000.2818047.

[17] H. R. Zeidanloo and A. A. Manaf, "Botnet command and control mechanisms," in *2009 Second International Conference on Computer and Electrical Engineering*, vol. 1, 2009, pp. 564–568. DOI: 10.1109/ICCEE.2009.151.

[18] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007. DOI: 10.1109/COMST.2007.4317620.

[19] M. Casenove and A. Miraglia, "Botnet over tor: The illusion of hiding," in *2014 6th International Conference On Cyber Conflict (CyCon 2014)*, 2014, pp. 273–282. DOI: 10.1109/CYCON.2014.6916408.

[20] C. Marrison, "Understanding the threats to dns and how to secure it," *Network Security*, vol. 2015, no. 10, pp. 8–10, 2015, ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(15)30090-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1353485815300908.

[21] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. v. Steen, and N. Pohlmann, "On botnets that use dns for command and control," in *2011 Seventh European Conference on Computer Network Defense*, 2011, pp. 9–16. DOI: 10.1109/EC2ND.2011.16.

[22] S. Yassine, J. Khalife, M. Chamoun, and H. El Ghor, "A survey of dns tunnelling detection techniques using machine learning.," in *BDCSIntell*, 2018, pp. 63–66.

[23] A. Ramdas and R. Muthukrishnan, "A survey on dns security issues and mitigation techniques," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, IEEE, 2019, pp. 781–784.

[24] Cloudflare. "What are the different types of dns server?" (2022), [Online]. Available: https://www.cloudflare.com/learning/dns/dns-server-types/.

[25] IBM. "Domain name system resource records." (2021), [Online]. Available: https://www.ibm.com/docs/en/i/7.1?topic=concepts-dns-resource-records.

[26] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols.* addison-Wesley, 2011.

[27] S. Shafieian, D. Smith, and M. Zulkernine, "Detecting dns tunneling using ensemble learning," in *Network and System Security*, Z. Yan, R. Molva, W. Mazurczyk, and R. Kantola, Eds., Cham: Springer International Publishing, 2017, pp. 112–127, ISBN: 978-3-319-64701-2.

[28] N. Raghava, D. Sahgal, and S. Chandna, "Classification of botnet detection based on botnet architechture," in *2012 International Conference on Communication Systems and Network Technologies*, 2012, pp. 569–572. DOI: 10.1109/CSNT.2012.128.

[29] Y. Xing, H. Shu, H. Zhao, D. Li, and L. Guo, "Survey on botnet detection techniques: Classification, methods, and evaluation," *Mathematical Problems in Engineering*, vol. 2021, 2021.

[30] H. R. Zeidanloo, M. J. Z. Shooshtari, P. V. Amoli, M. Safari, and M. Zamani, "A taxonomy of botnet detection techniques," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 2, 2010, pp. 158–162. DOI: 10.1109/ICCSIT.2010.5563555.

[31] M. Singh, M. Singh, and S. Kaur, "Issues and challenges in dns based botnet detection: A survey," *Computers & Security*, vol. 86, pp. 28–52, 2019.

[32] Sudhakar and S. Kumar, "Botnet detection techniques and research challenges," in *2019 International Conference on Recent Advances in Energy-efficient Computing and Communication (ICRAECC)*, 2019, pp. 1–6. DOI: `10.1109/ICRAECC43874.2019.8995028`.

[33] Y. Meidan, M. Bohadana, Y. Mathov, *et al.*, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018. DOI: `10.1109/MPRV.2018.03367731`.

[34] R. P. Padhy, M. R. Patra, and S. C. Satapathy, "Cloud computing: Security issues and research challenges," *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, vol. 1, no. 2, pp. 136–146, 2011.

[35] A. H. Lashkari, G. D. Gil, J. E. Keenan, K. F. Mbah, and A. A. Ghorbani, "A survey leading to a new evaluation framework for network-based botnet detection," in *Proceedings of the 2017 the 7th International Conference on Communication and Network Security*, ser. ICCNS 2017, Tokyo, Japan: Association for Computing Machinery, 2017, pp. 59–66, ISBN: 9781450353496. DOI: `10.1145/3163058.3163059`. [Online]. Available: `https://doi.org/10.1145/3163058.3163059`.

[36] D. Hein. "Active monitoring and passive monitoring: What's the difference?" (2019), [Online]. Available: `https://solutionsreview.com/network-monitoring/active-monitoring-and-passive-monitoring-whats-the-difference/`.

[37] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016, ISSN: 1084-8045. DOI: `https://doi.org/10.1016/j.jnca.2015.11.016`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1084804515002891`.

[38] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, 4. Springer, 2006, vol. 4.

[39] "The science: Orbital mechanics." (2009), [Online]. Available: `https://earthobservatory.nasa.gov/features/OrbitsHistory/page2.php`.

[40] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona, "A survey on missing data in machine learning," *Journal of Big Data*, vol. 8, no. 1, pp. 1–37, 2021.

[41] S. Garcıa, S. Ramırez-Gallego, J. Luengo, J. M. Benıtez, and F. Herrera, "Big data preprocessing: Methods and prospects," *Big Data Analytics*, vol. 1, no. 1, pp. 1–22, 2016.

[42] S. A. Alasadi and W. S. Bhaya, "Review of data preprocessing techniques in data mining," *Journal of Engineering and Applied Sciences*, vol. 12, no. 16, pp. 4102–4107, 2017.

[43] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014, 40th-year commemorative issue, ISSN: 0045-7906. DOI: `https://doi.org/10.1016/j.compeleceng.2013.11.024`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0045790613003066`.

[44] T. M. Cover, J. A. Thomas, *et al.*, "Entropy, relative entropy and mutual information," *Elements of information theory*, vol. 2, no. 1, pp. 12–13, 1991.

[45] G. Forman *et al.*, "An extensive empirical study of feature selection metrics for text classification.," *J. Mach. Learn. Res.*, vol. 3, no. Mar, pp. 1289–1305, 2003.

[46] "Recursive feature elimination (rfe) for feature selection in python." (2020), [Online]. Available: `https://machinelearningmastery.com/rfe-feature-selection-in-python/`.

[47] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 1200–1205. DOI: `10.1109/MIPRO.2015.7160458`.

[48] V. Fonti and E. Belitser, "Feature selection using lasso," *VU Amsterdam research paper in business analytics*, vol. 30, pp. 1–25, 2017.

[49] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[50] R. Zebari, A. Abdulazeez, D. Zeebaree, D. Zebari, and J. Saeed, "A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction," *Journal of Applied Science and Technology Trends*, vol. 1, no. 2, pp. 56–70, 2020.

[51] V. Spruyt. "The curse of dimensionality in classification." (2014), [Online]. Available: `https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/`.

[52] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[53] J. Jauregui, "Principal component analysis with linear algebra," *Philadelphia: Penn Arts & Sciences*, 2012.

[54] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[55] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, *et al.*, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 756–795, 2021.

[56] H. Hojjati, T. K. K. Ho, and N. Armanfard, "Self-supervised anomaly detection: A survey and outlook," *arXiv preprint arXiv:2205.05173*, 2022.

[57] Scikit-Learn. "Novelty and outlier detection." (2022), [Online]. Available: `https://scikit-learn.org/stable/modules/outlier_detection.html`.

[58] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal processing*, vol. 99, pp. 215–249, 2014.

[59] S. W. Smith, "Properties of convolution," *The Scientist and Engineer's Guide to Digital Signal Processing*, pp. 123–140, 1997.

[60] D. M. Johannes, "One-class classification," Ph.D. dissertation, Delft University of Technology, 2001.

[61] D. Miljković, "Review of novelty detection methods," in *The 33rd International Convention MIPRO*, IEEE, 2010, pp. 593–598.

[62] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[63] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," *Advances in neural information processing systems*, vol. 12, 1999.

[64] M. R. Oliveira, J. Neves, R. Valadas, and P. Salvador, "Do we need a perfect ground-truth for benchmarking internet traffic classifiers?" In *2015 IEEE Conference on Computer Communications (INFOCOM)*, IEEE, 2015, pp. 2452–2460.

[65] L. Böck, M. Fejrskov, K. Demetzou, S. Karuppayah, M. Mühlhäuser, and E. Vasilomanolakis, "Processing of botnet tracking data under the gdpr," *Computer Law & Security Review*, vol. 45, p. 105 652, 2022.

[66] A. Merlo, G. Papaleo, S. Veneziano, and M. Aiello, “A comparative performance evaluation of dns tunneling tools,” in *Computational Intelligence in Security for Information Systems*, Springer, 2011, pp. 84–91.

[67] Y. Wang, A. Zhou, S. Liao, R. Zheng, R. Hu, and L. Zhang, “A comprehensive survey on dns tunnel detection,” *Computer Networks*, vol. 197, p. 108 322, 2021.

[68] P. Wang, L. Wu, B. Aslam, and C. C. Zou, “A systematic study on peer-to-peer botnets,” in *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, IEEE, 2009, pp. 1–8.

[69] B. Choi, S.-K. Choi, and K. Cho, “Detection of mobile botnet using vpn,” in *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IEEE, 2013, pp. 142–148.

[70] G. Gu, J. Zhang, and W. Lee, “Botsniffer: Detecting botnet command and control channels in network traffic,” 2008.

[71] E. Adi, A. Anwar, Z. Baig, and S. Zeadally, “Machine learning and data analytics for the iot,” *Neural Computing and Applications*, vol. 32, no. 20, pp. 16 205–16 233, 2020.

[72] Azelcast. “Sliding window aggregation.” (2022), [Online]. Available: `https://docs.hazelcast.com/hazelcast/5.1/architecture/sliding-window`.

[73] A. H. Lashkari, G. D. Gil, J. E. Keenan, K. F. Mbah, and A. A. Ghorbani, “A survey leading to a new evaluation framework for network-based botnet detection,” in *Proceedings of the 2017 the 7th International Conference on Communication and Network Security*, 2017, pp. 59–66.

[74] C. Pascoal, M. R. De Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco, “Robust feature selection and robust PCA for internet traffic anomaly detection,” in *2012 Proceedings Ieee Infocom*, IEEE, 2012, pp. 1755–1763.

[75] P. J. Huber, “Robust statistics,” in *International encyclopedia of statistical science*, Springer, 2011, pp. 1248–1251.

Licit DNS traffic Throughput - Time Series

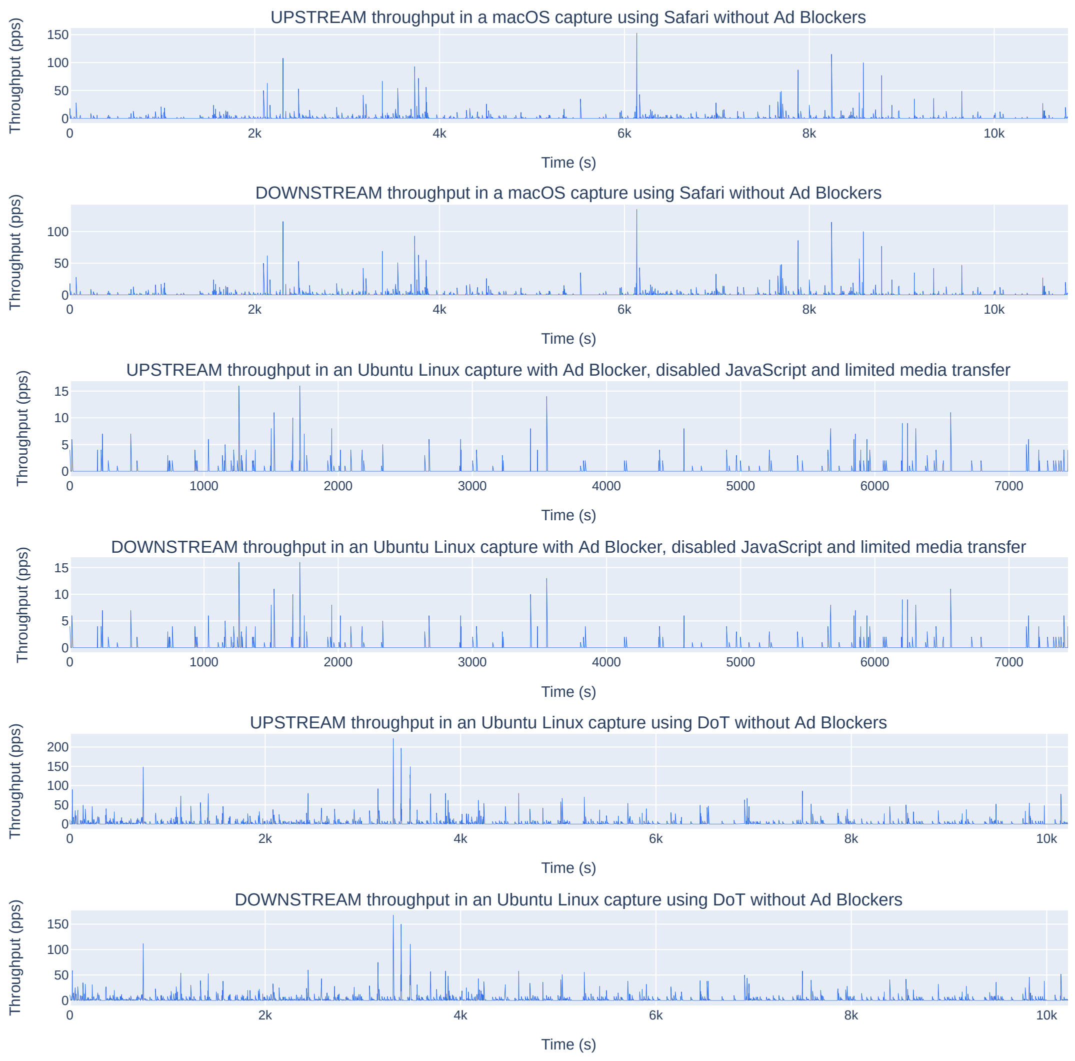


Figure 1: Examples of throughput in licit DNS traffic - Time Series

Licit DNS traffic - Histograms

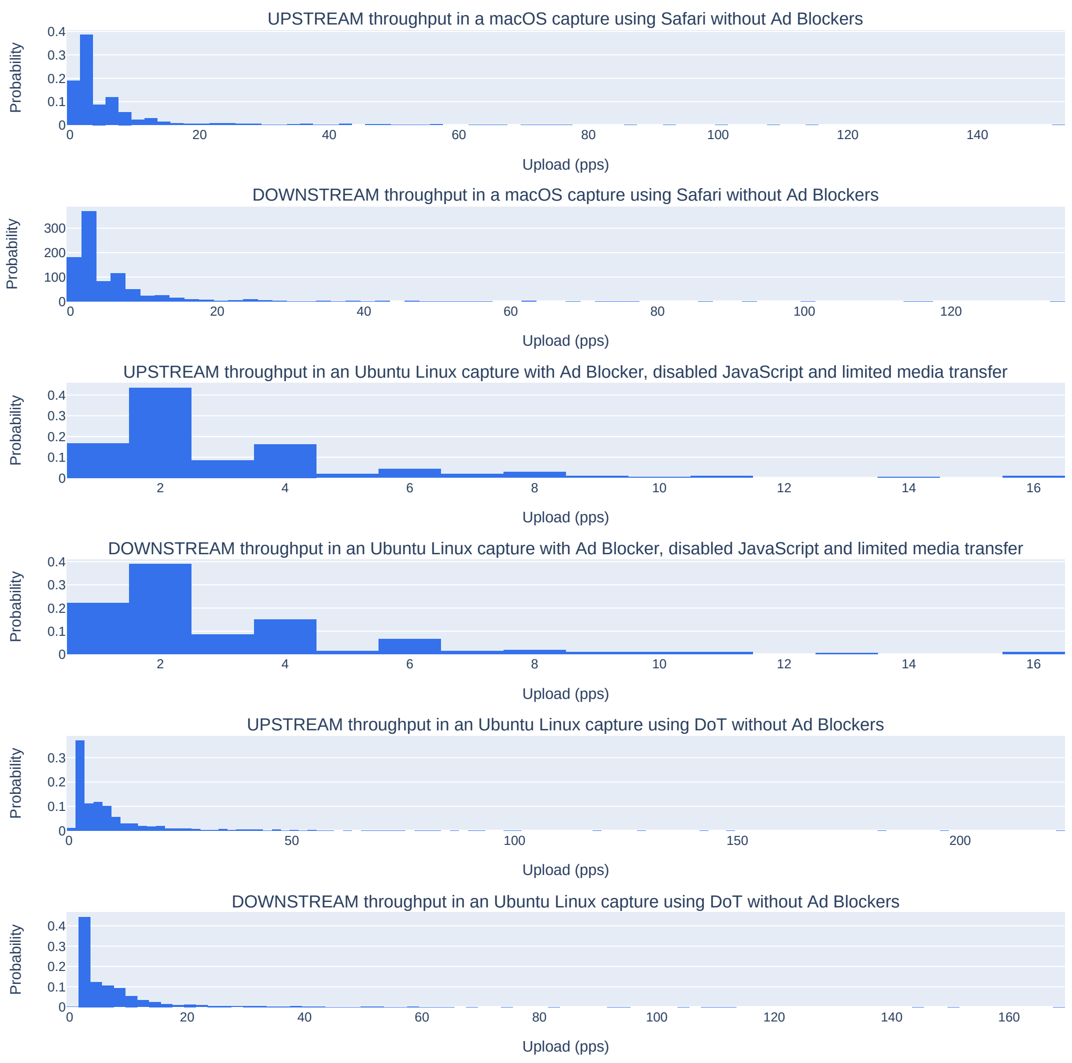


Figure 2: Examples of throughput in licit DNS traffic - Histogram

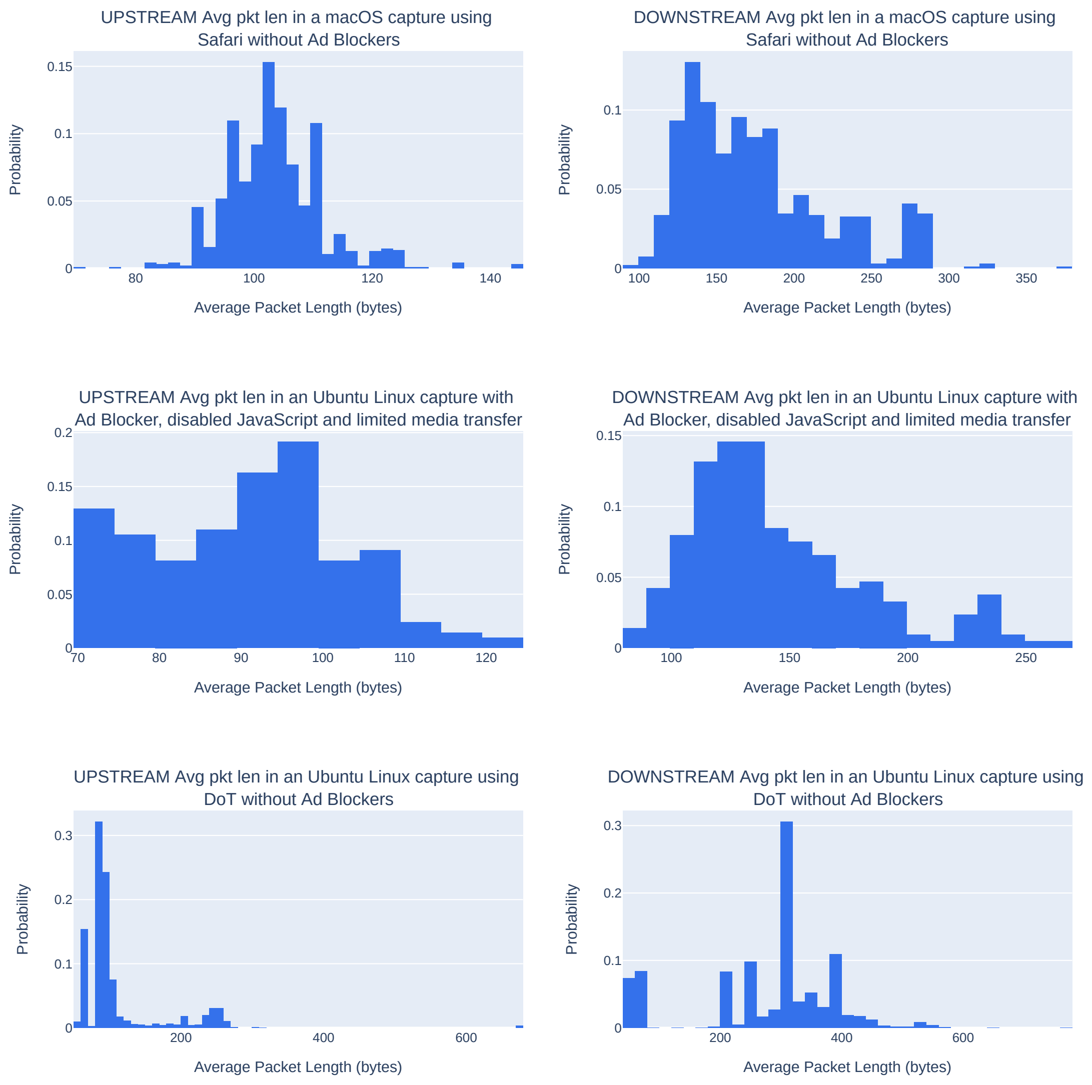


Figure 3: Examples of Average Packet Lengths in licit DNS traffic - Histogram

Scenario 1: Illicit DNS traffic - Histograms

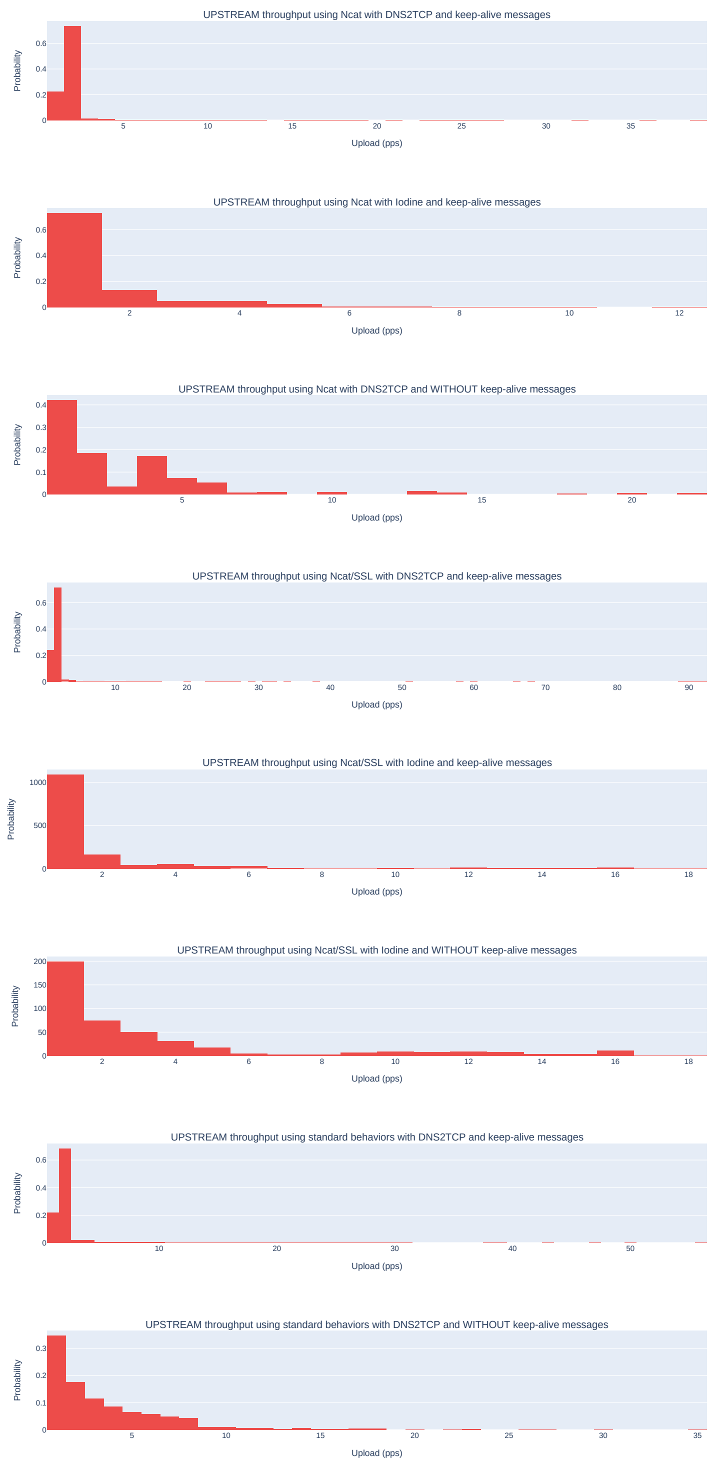


Figure 4: Examples of generated throughput for Scenario 1 - Histogram

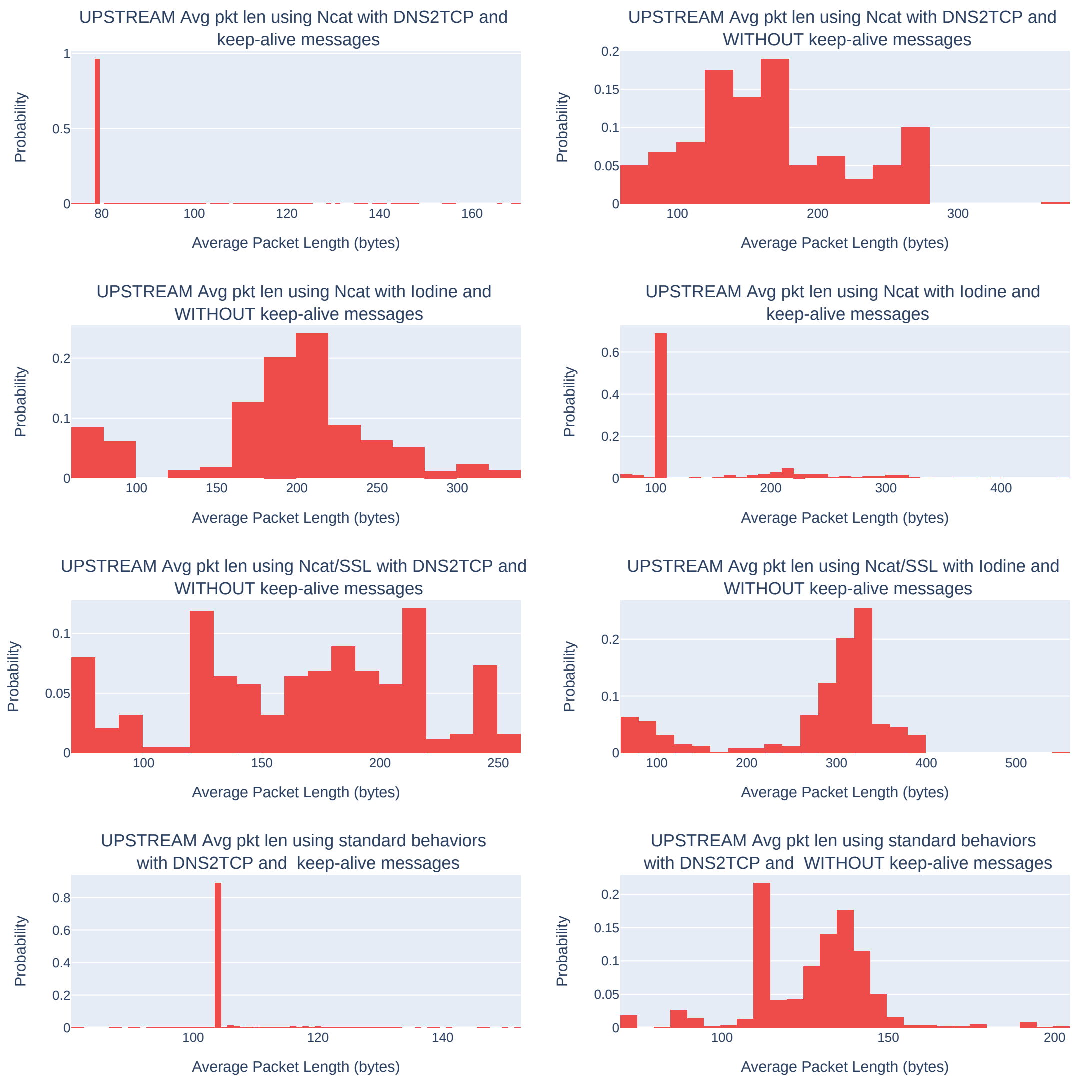


Figure 5: Examples of generated average packet length for Scenario 1 - Histogram

Scenario 3: Histogram comparison between licit and illicit traffic

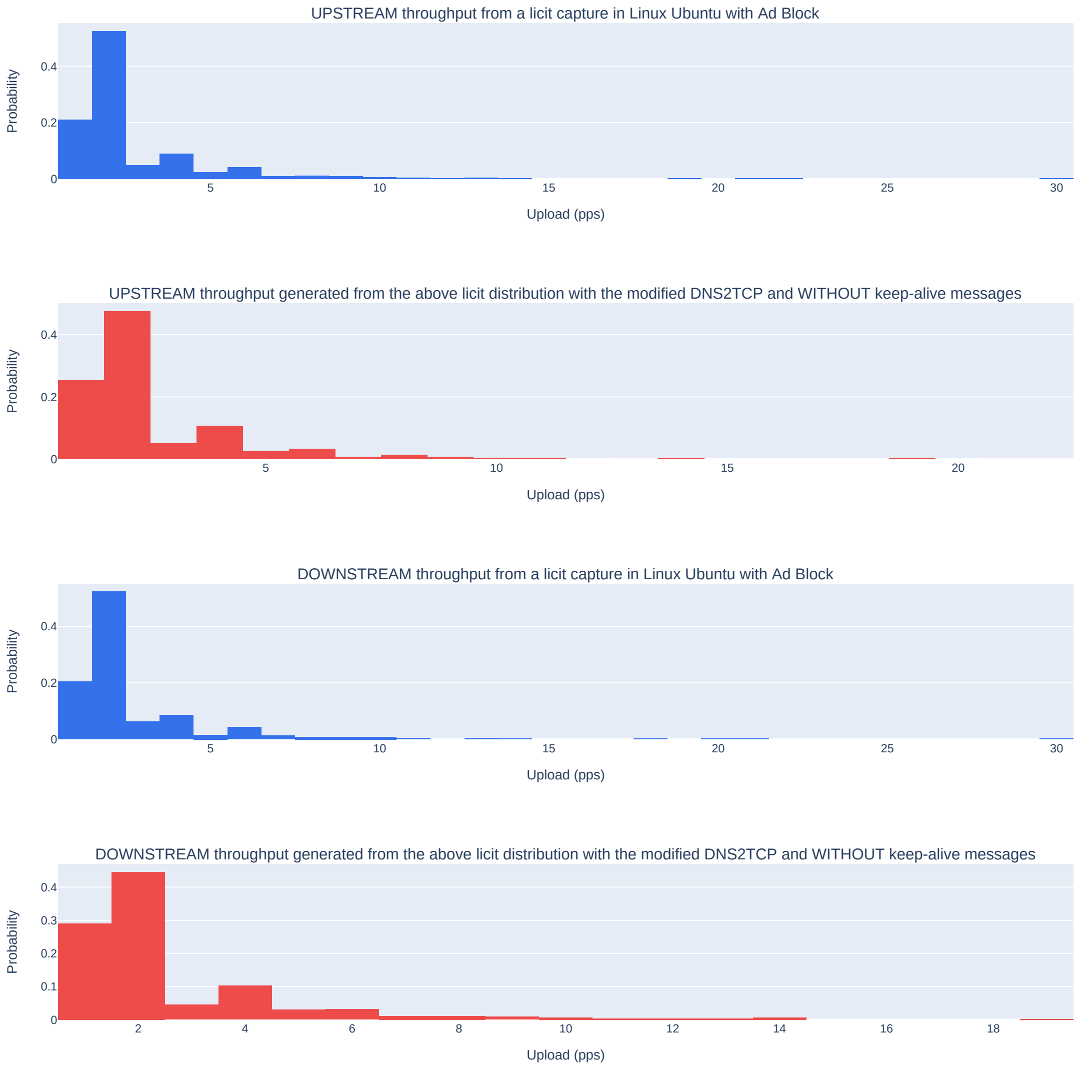


Figure 6: Comparison between licit (in blue) and illicit (in red) throughput generated by the modified DNS2TCP with $\alpha = 0.0$ for Scenario 3

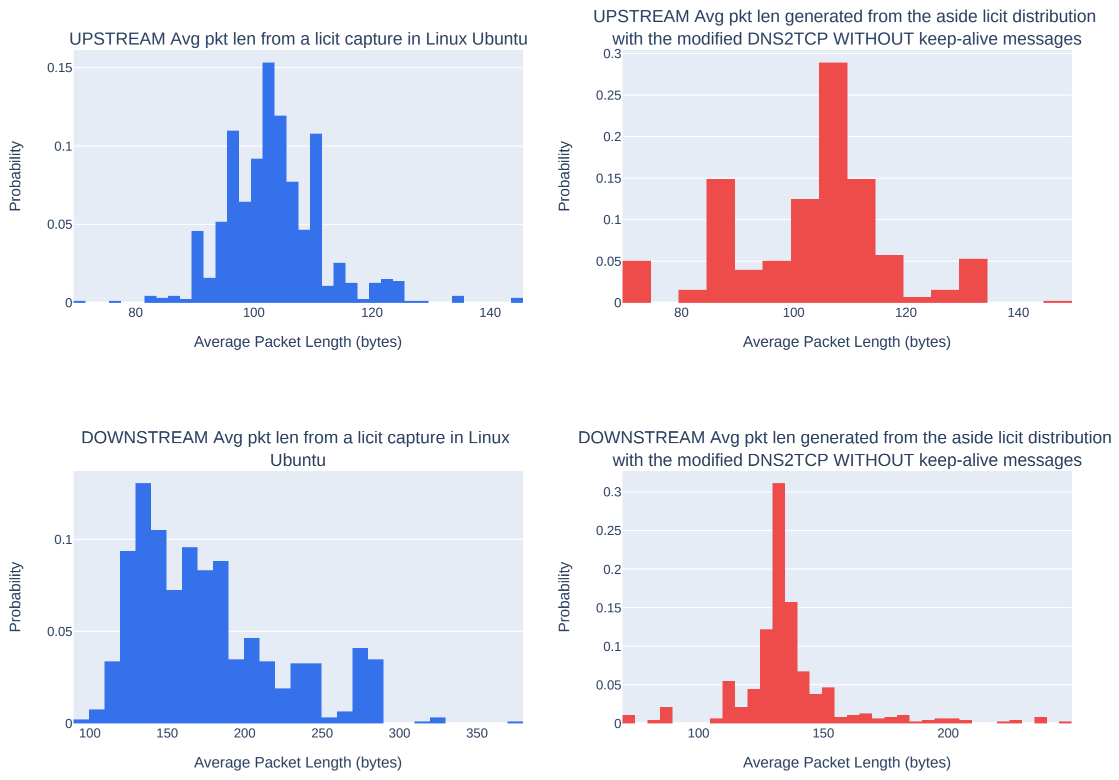


Figure 7: Comparison between licit (in blue) and illicit (in red) average packet lengths generated by the modified DNS2TCP with $\alpha = 0.0$ for Scenario 3

Dataset with a sliding window of 70 minutes

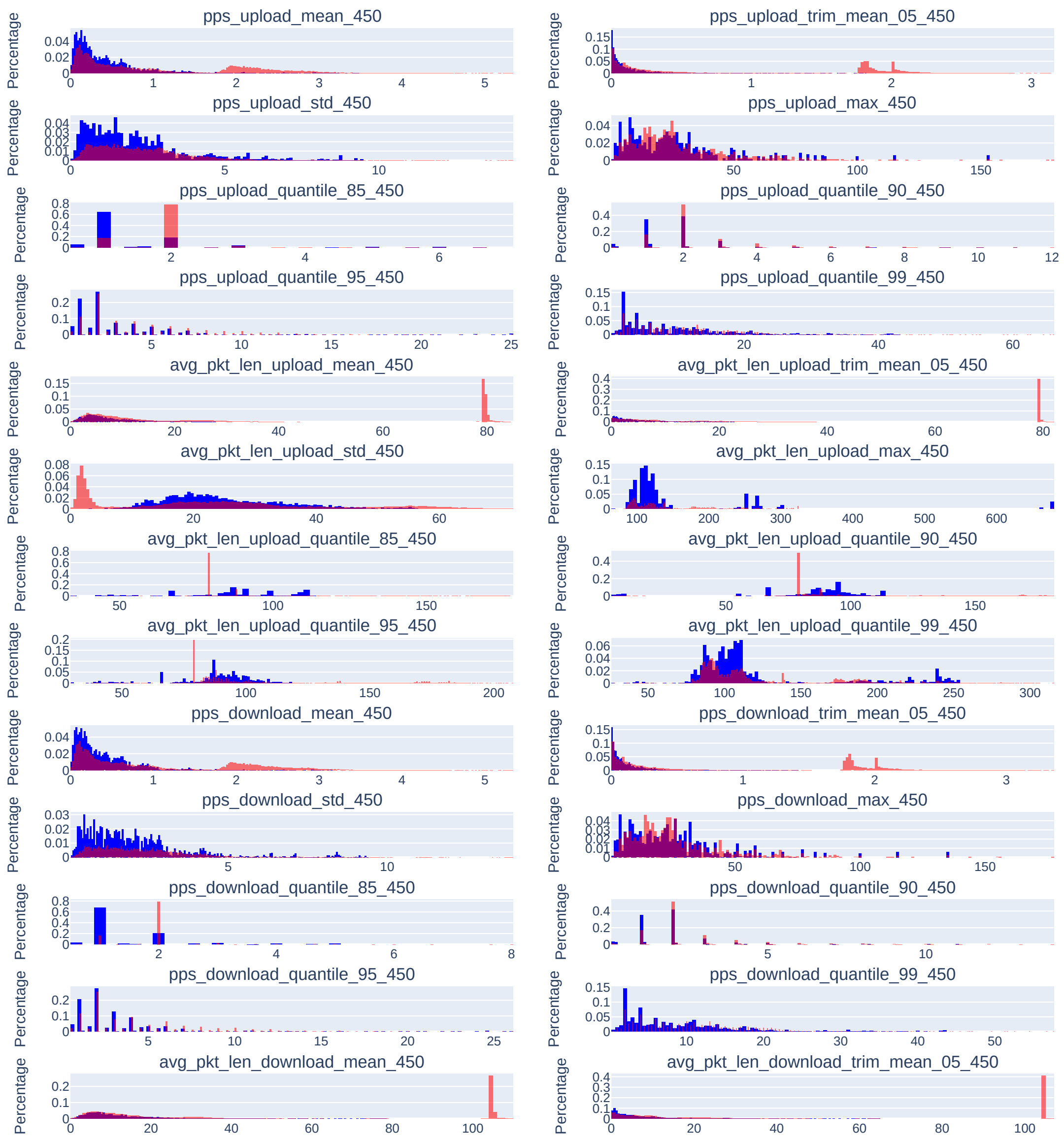


Figure 8: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 1

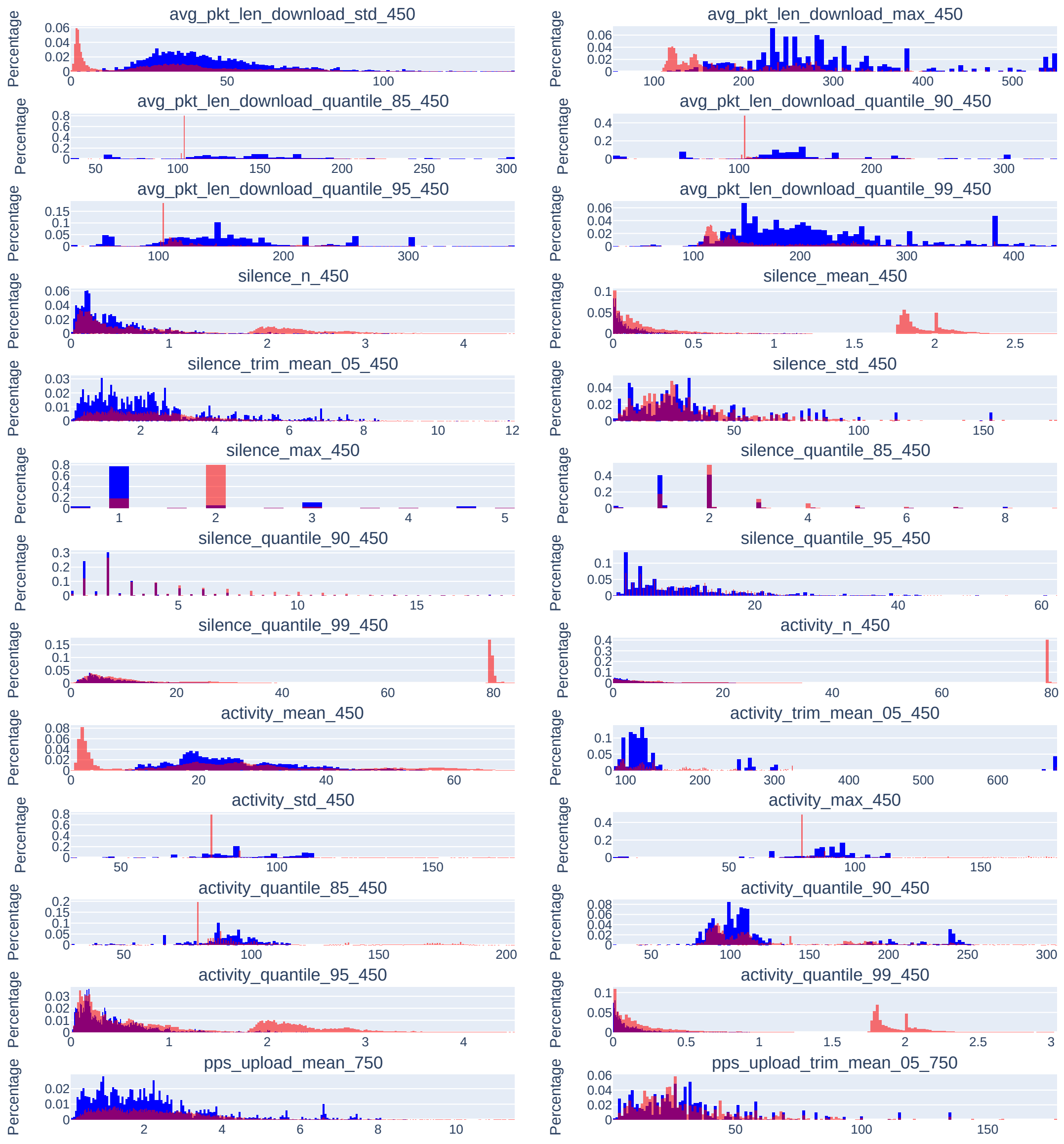


Figure 9: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 2

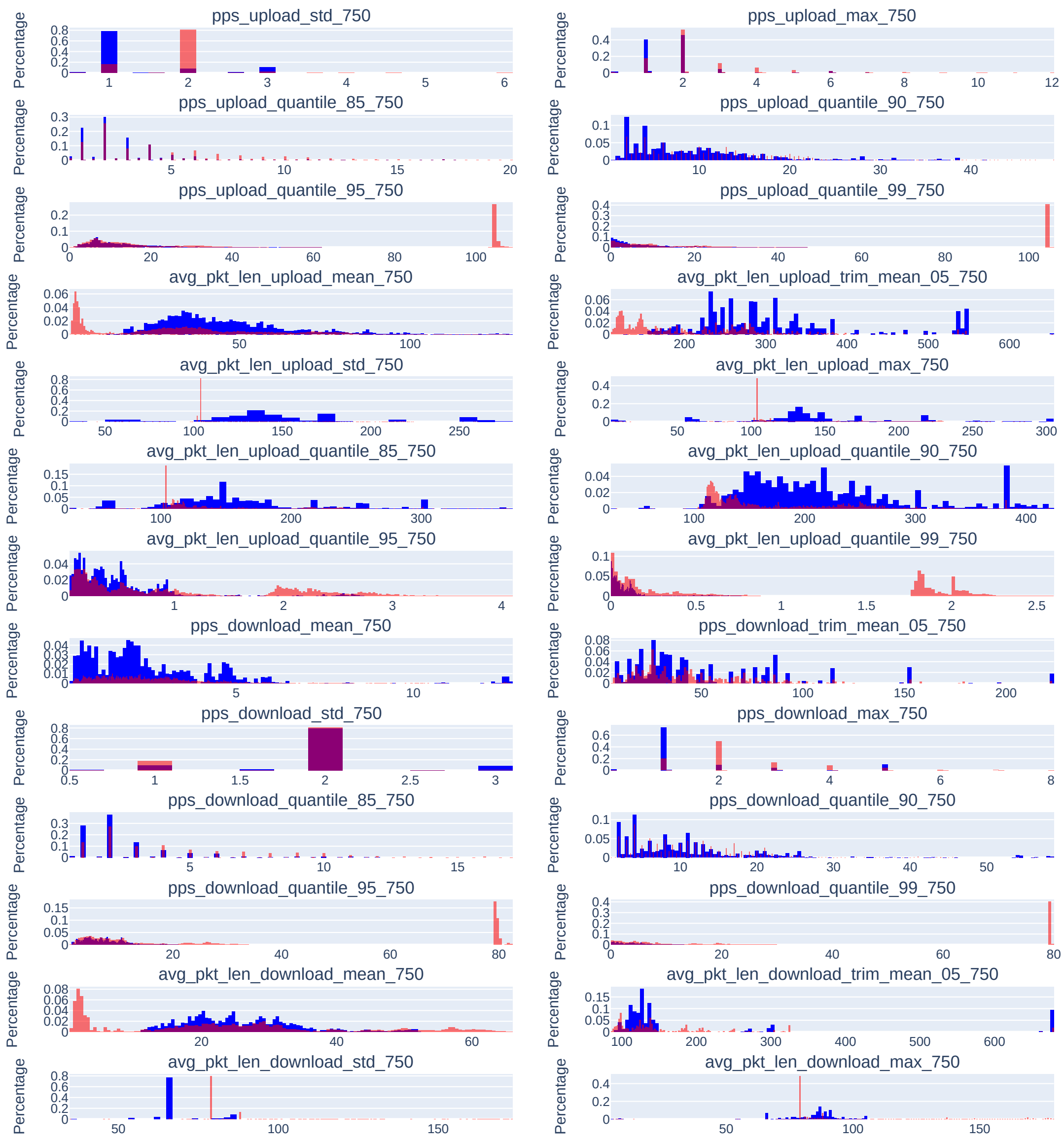


Figure 10: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 3

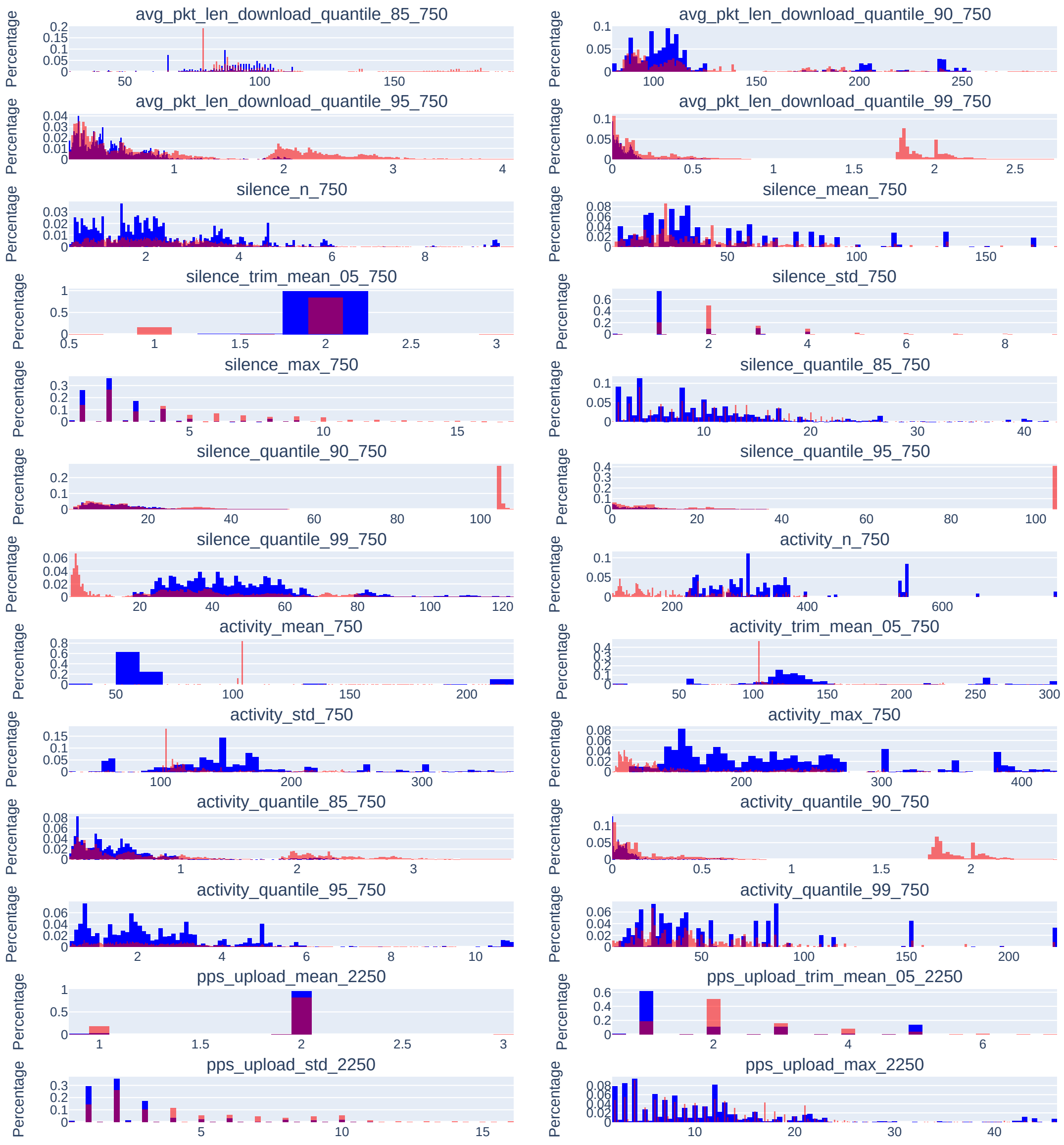


Figure 11: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 4

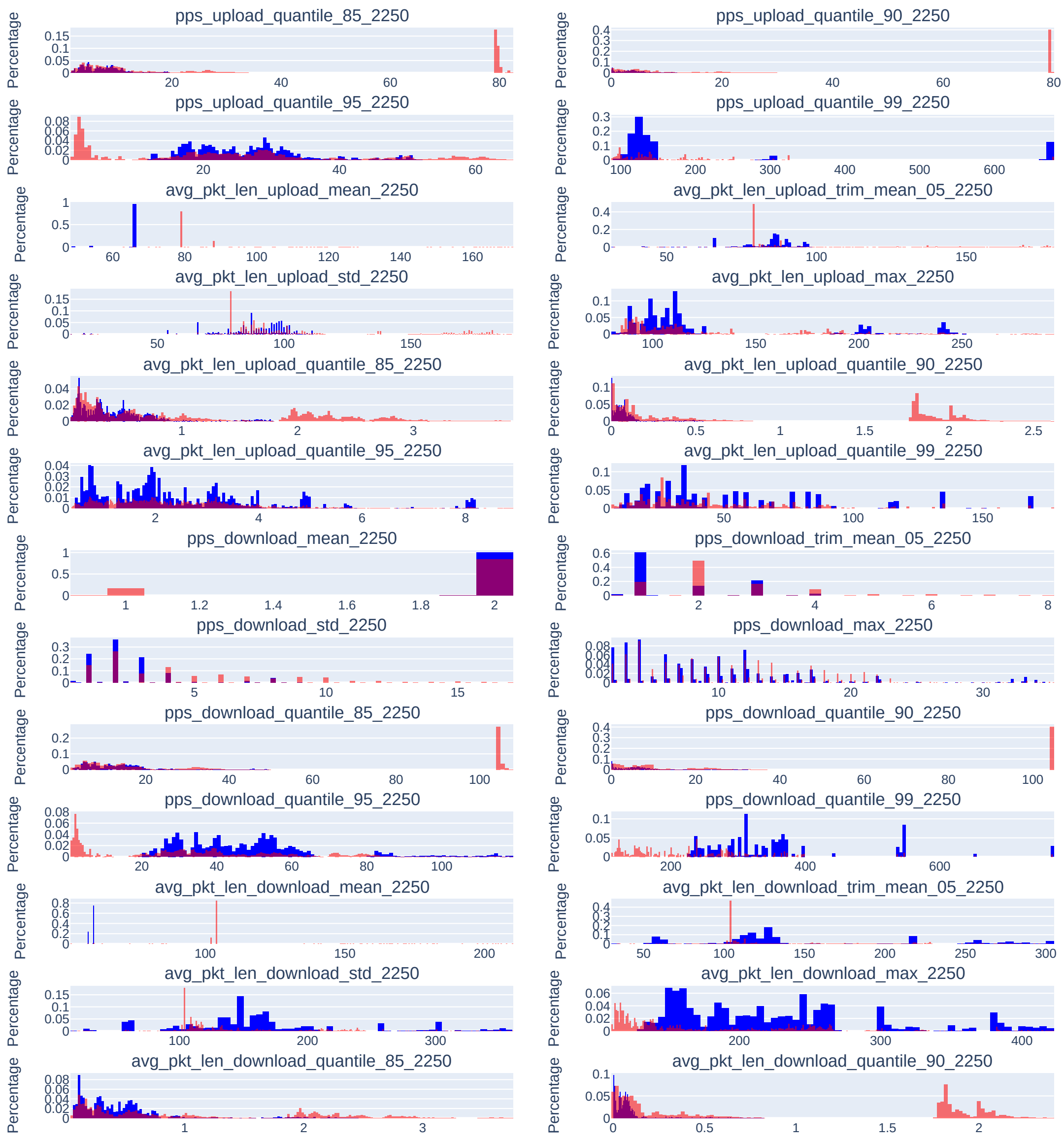


Figure 12: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 5

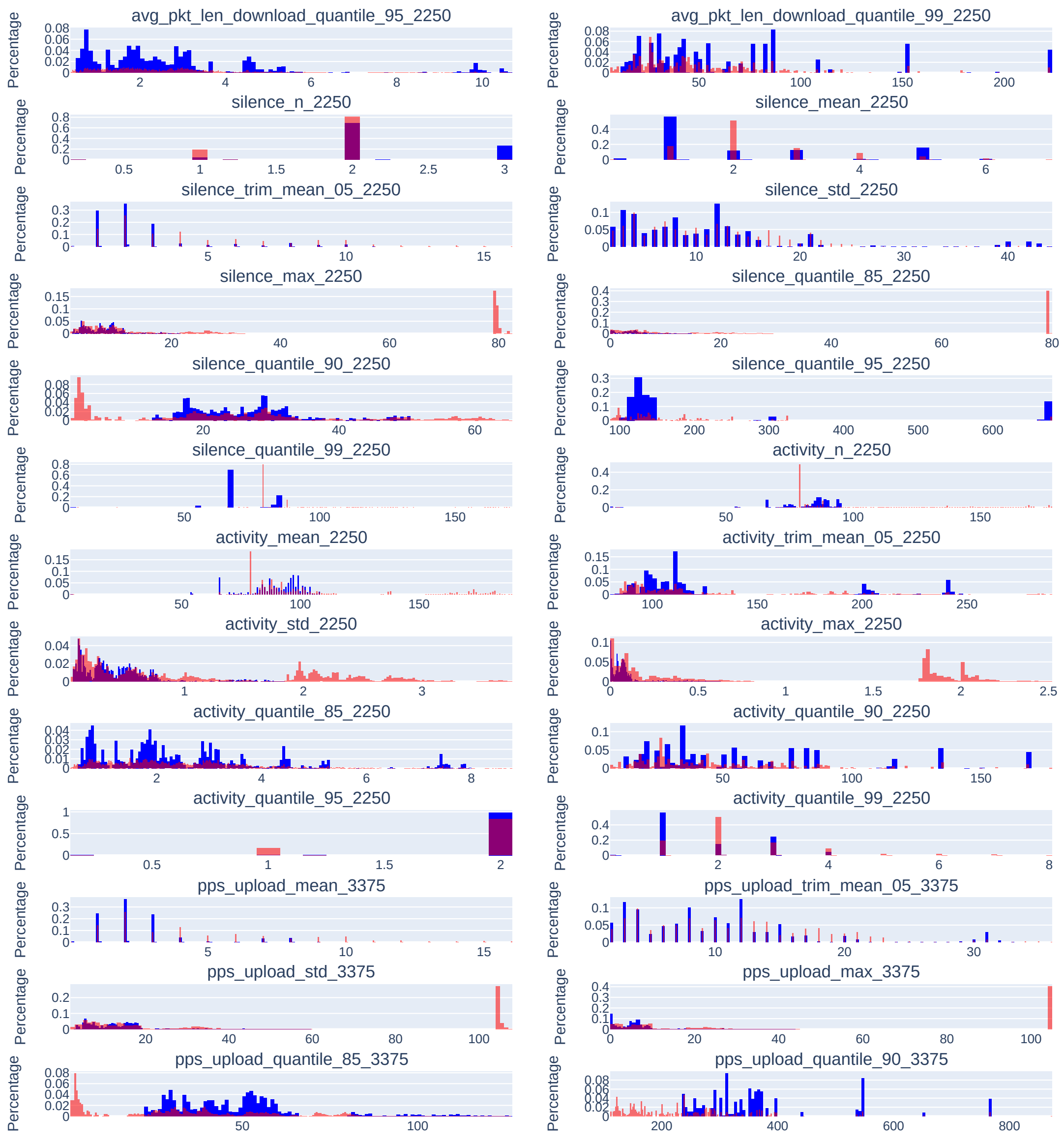


Figure 13: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 6

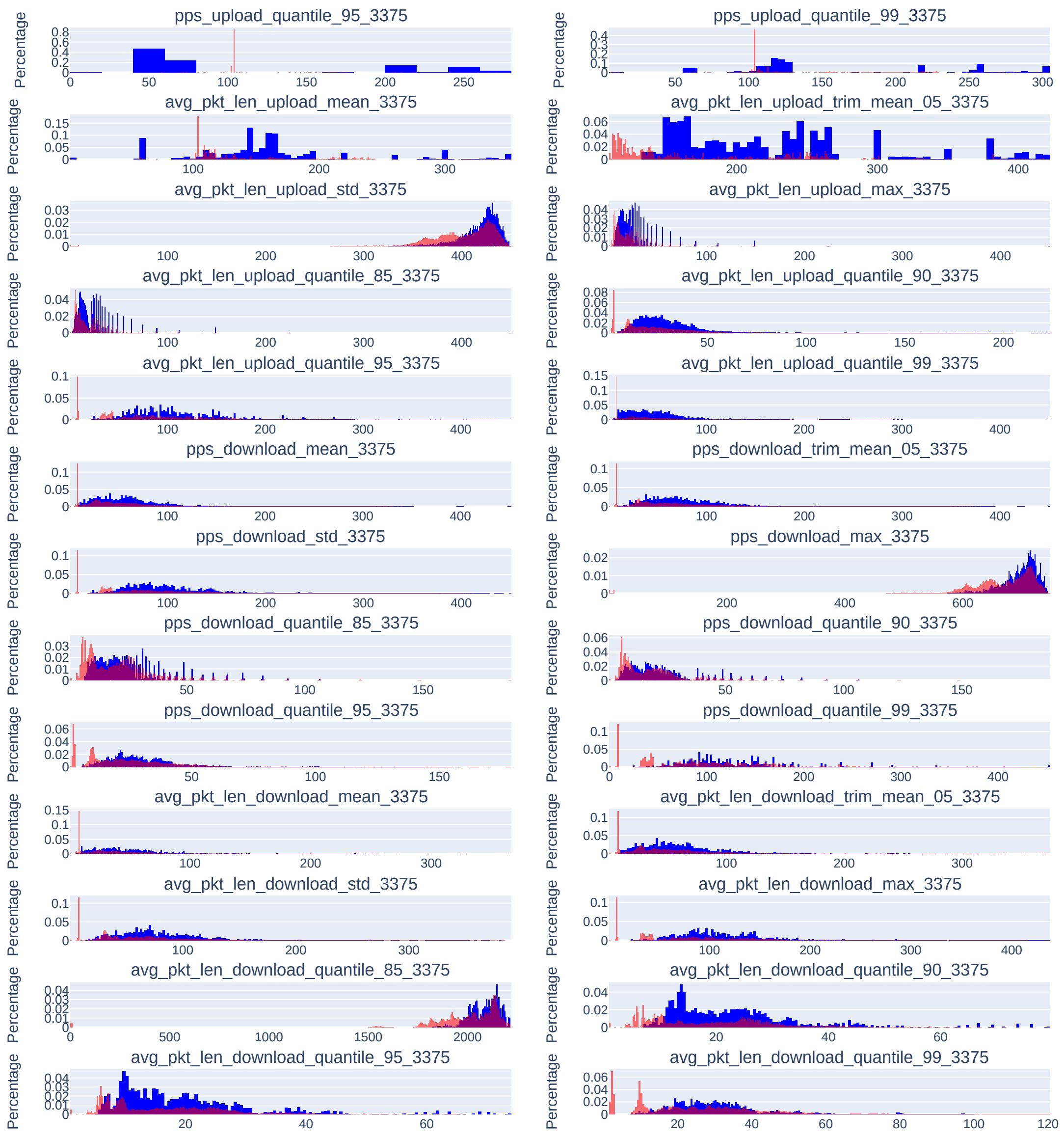


Figure 14: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 7

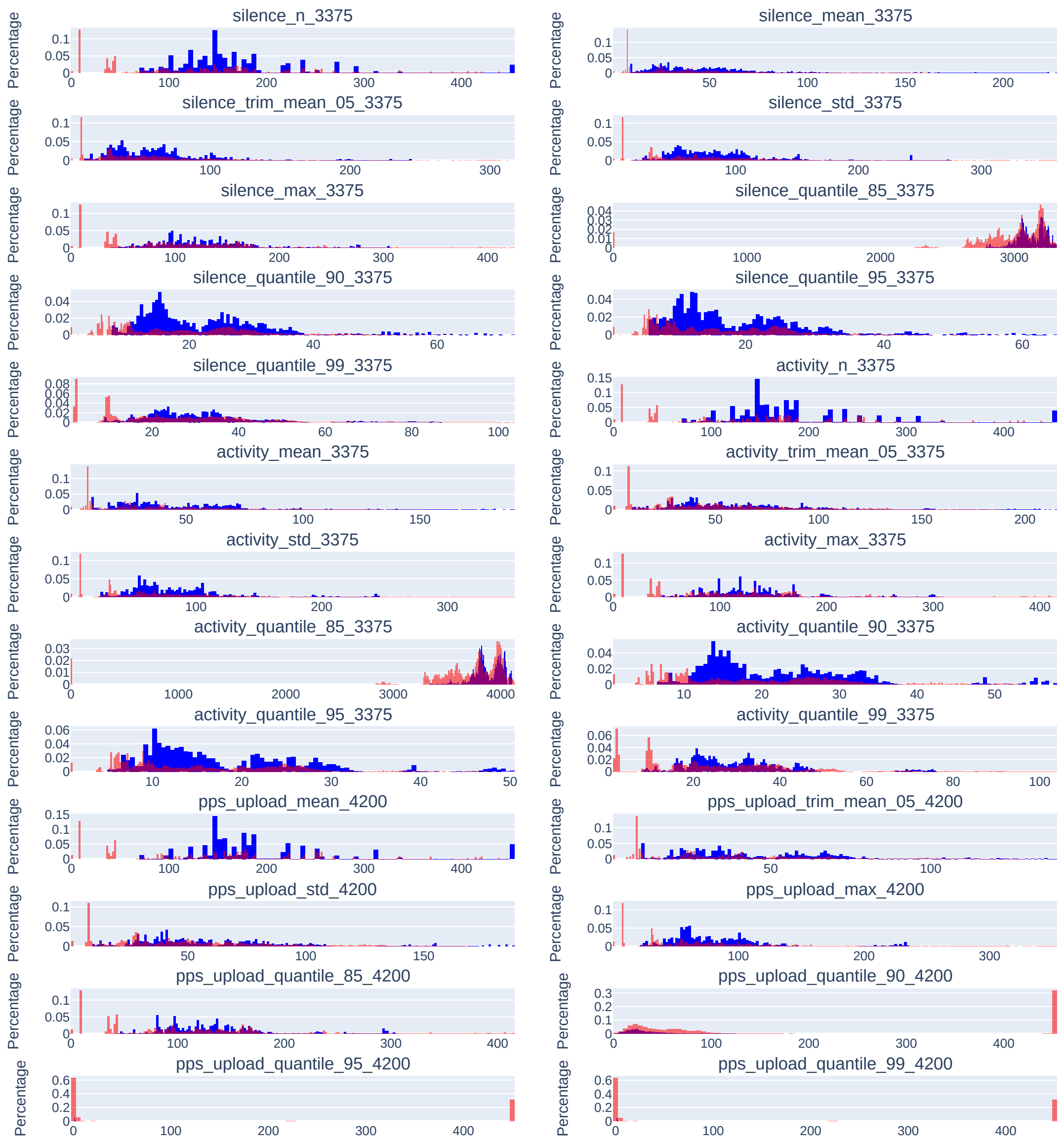


Figure 15: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 8

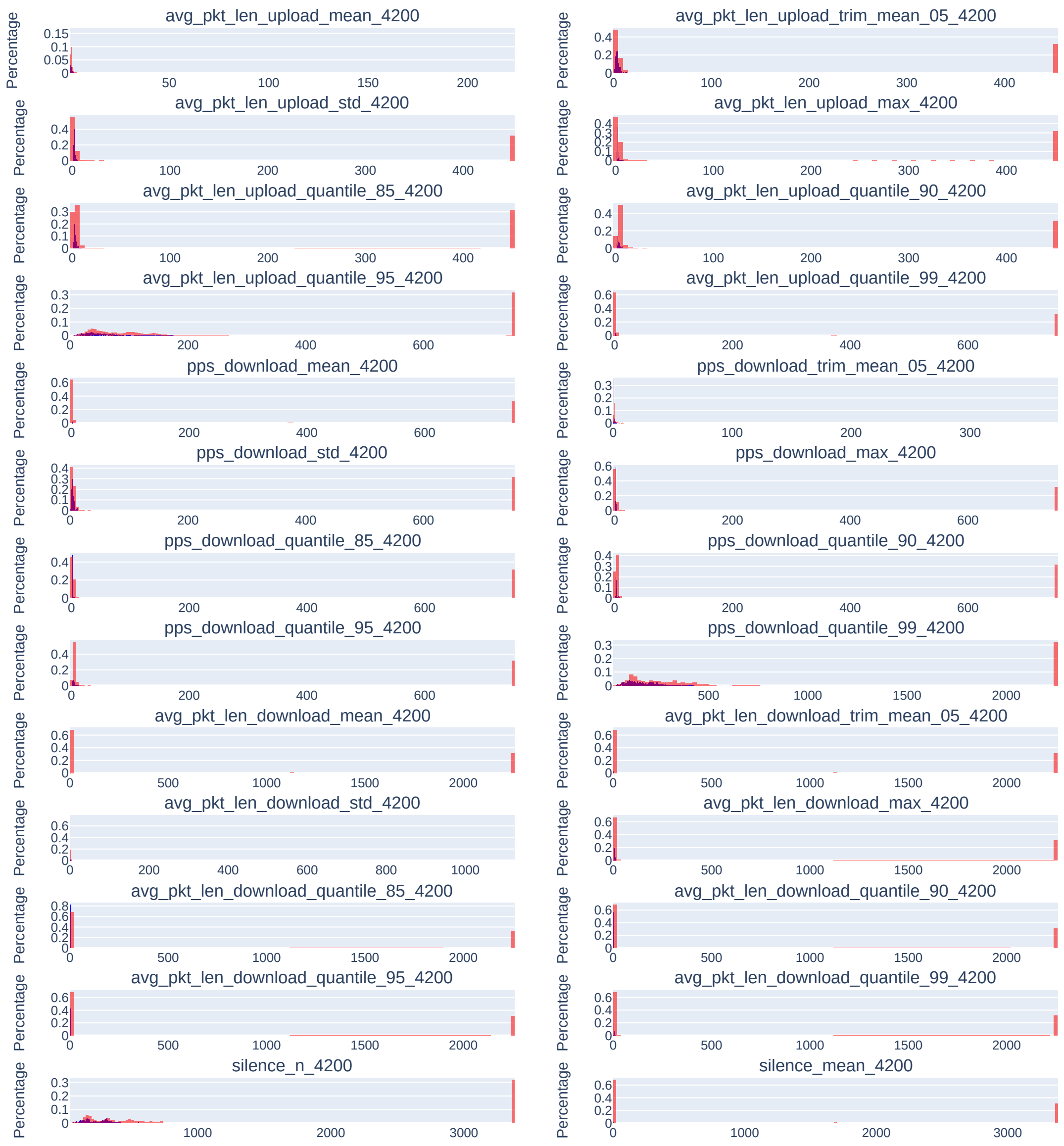


Figure 16: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 9

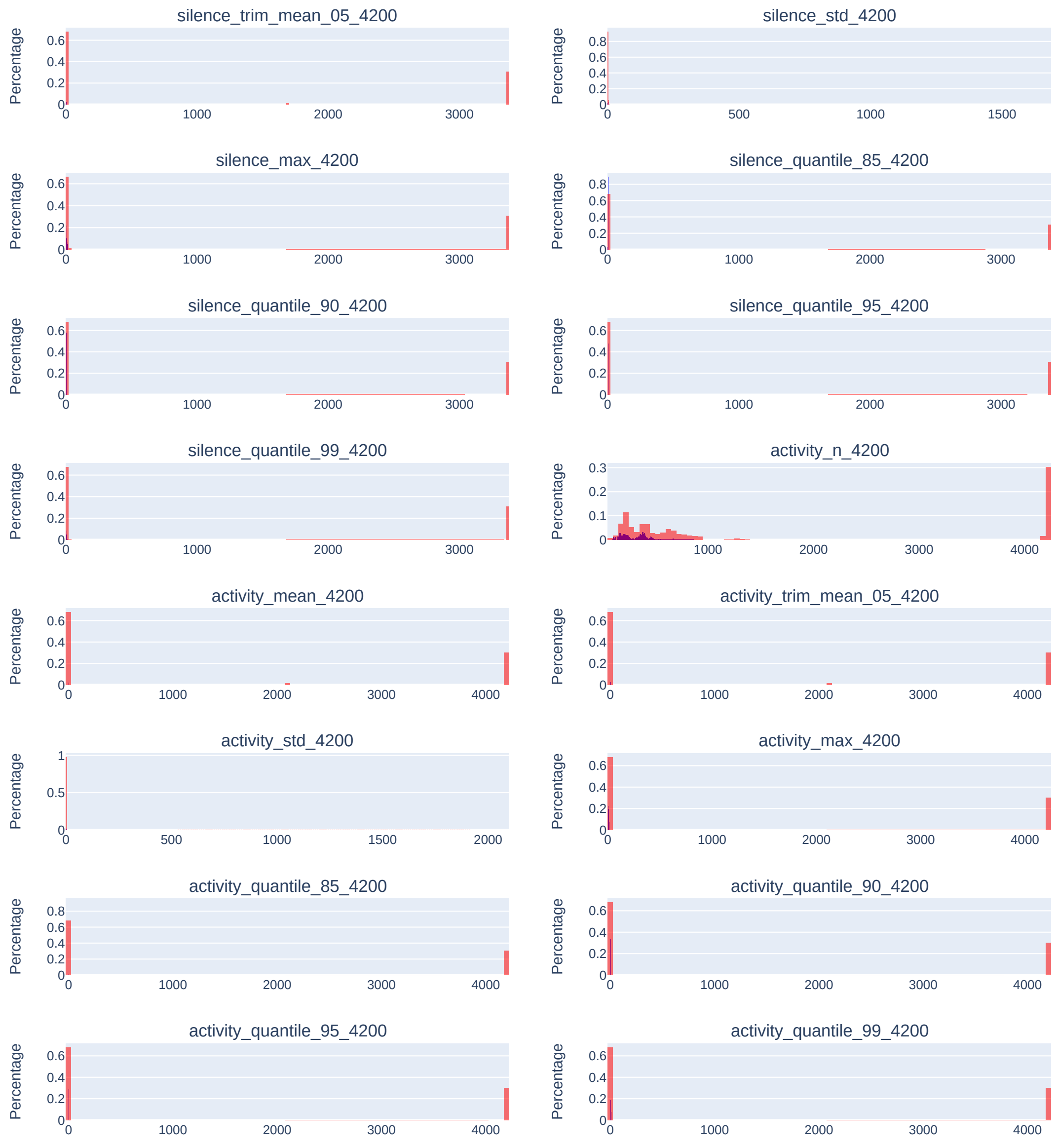


Figure 17: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 70 minutes - Part 10

Dataset with a sliding window of 15 minutes

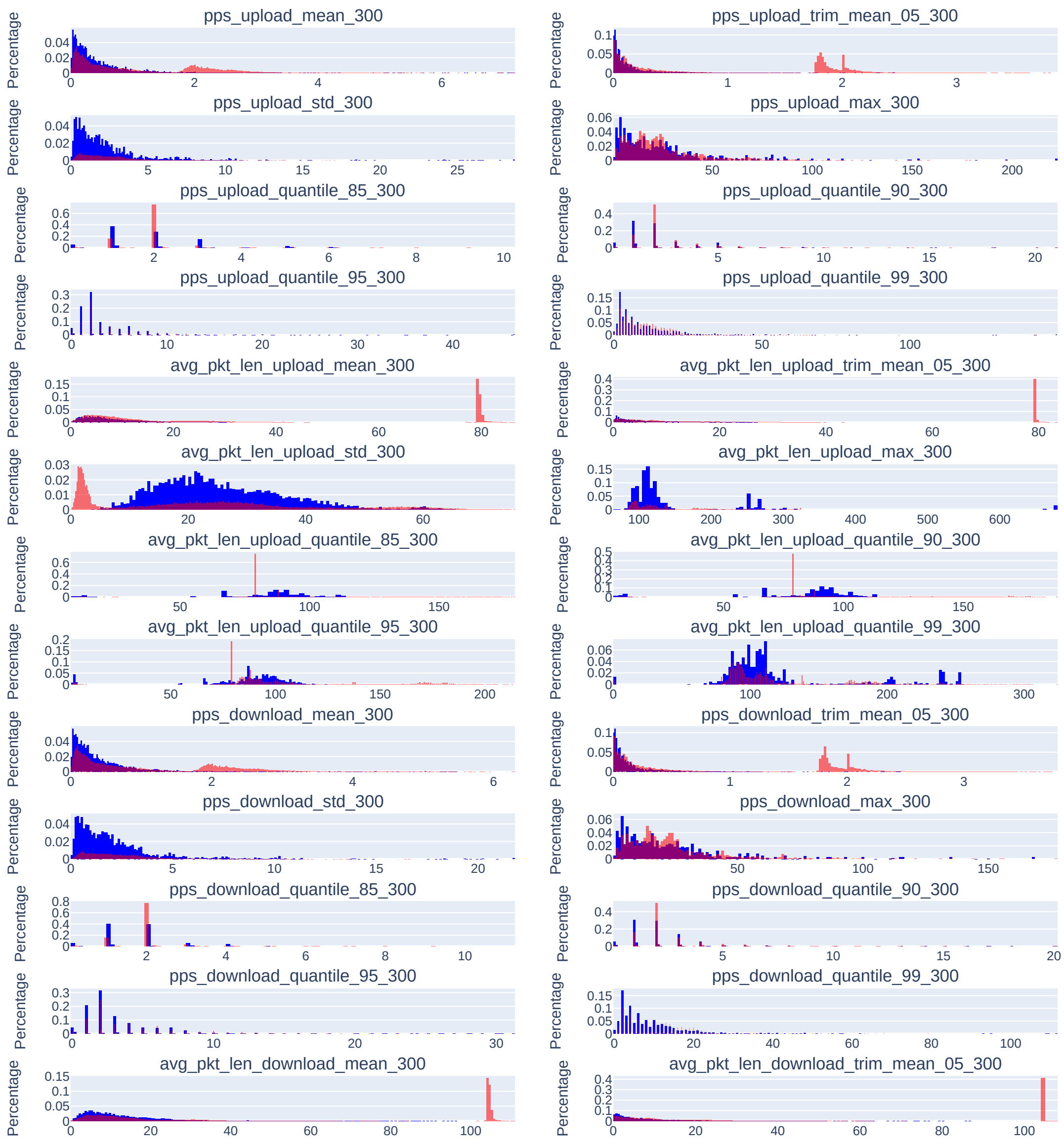


Figure 18: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 15 minutes - Part 1

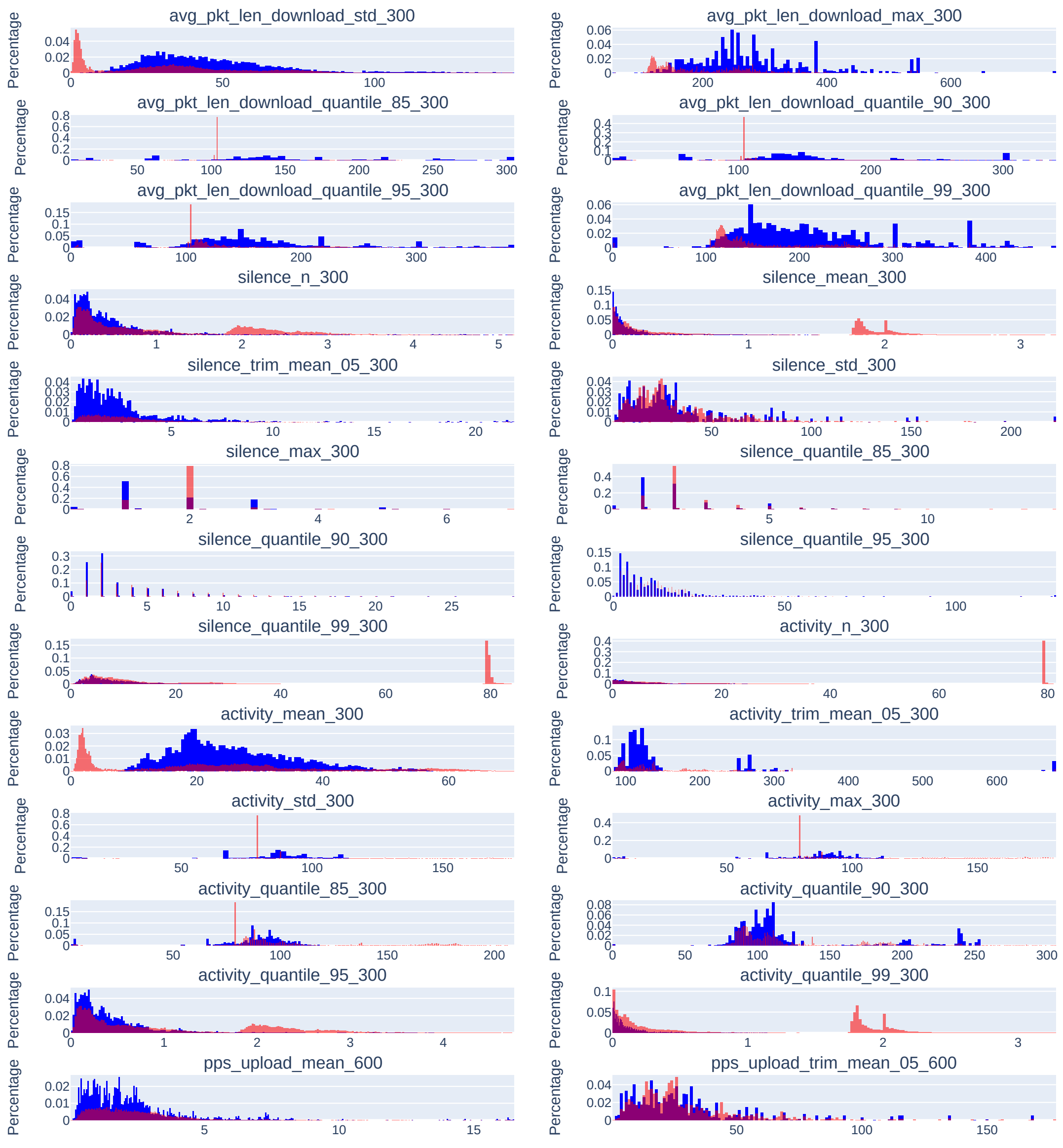


Figure 19: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 15 minutes - Part 2

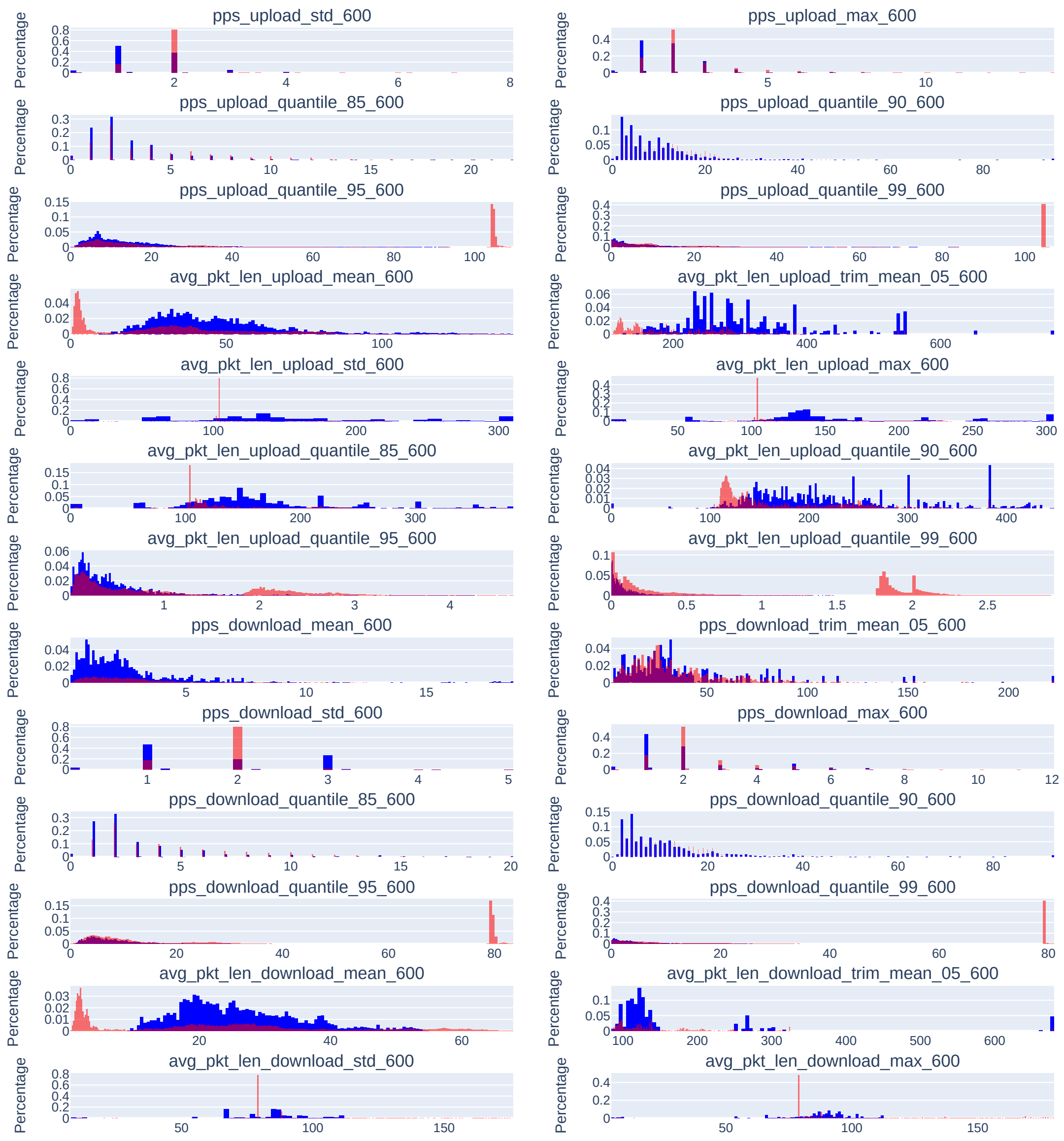


Figure 20: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 15 minutes - Part 3

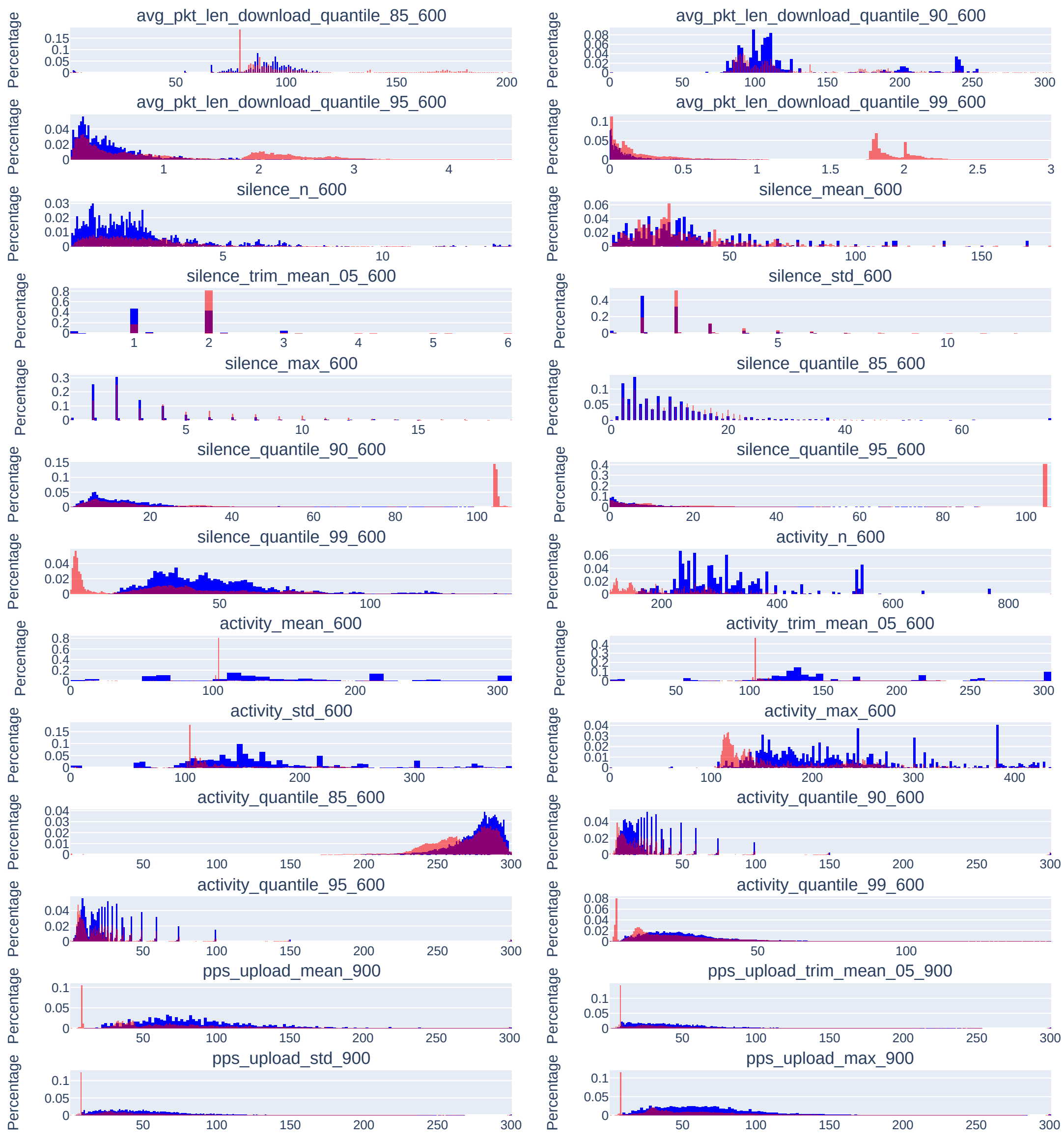


Figure 21: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 15 minutes - Part 4

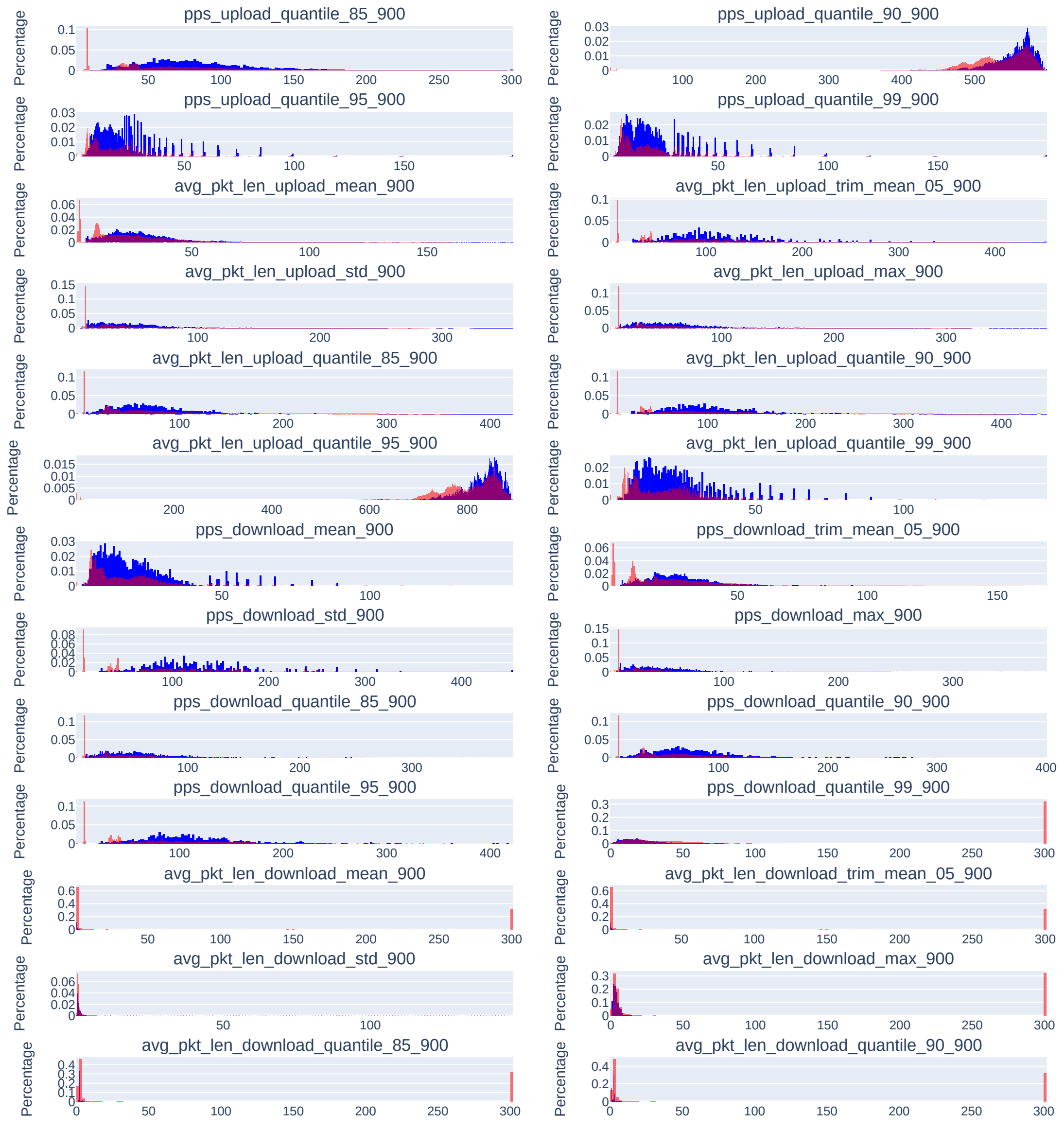


Figure 22: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 15 minutes - Part 5

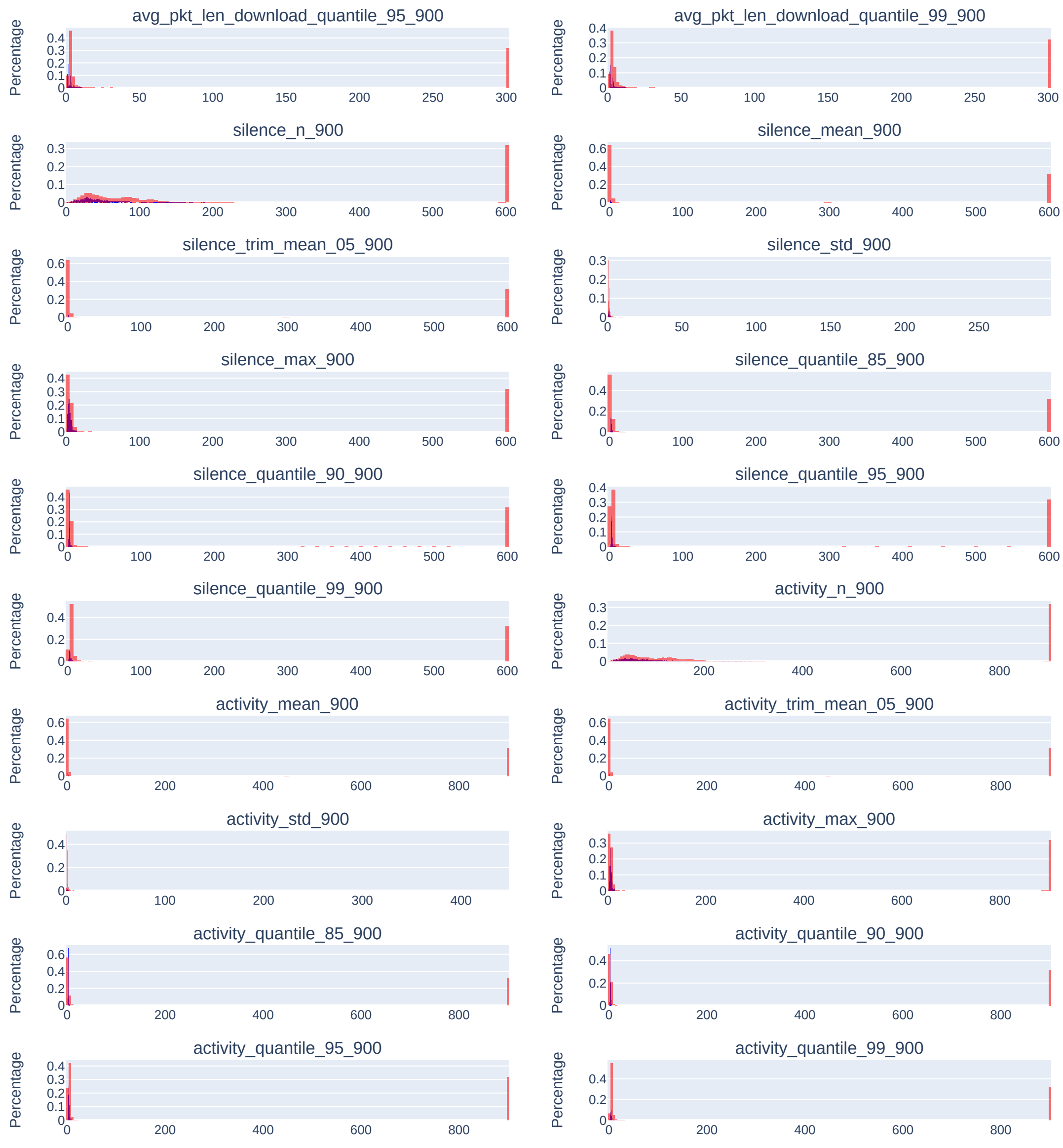


Figure 23: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 15 minutes - Part 6

Dataset with a sliding window of 75 minutes

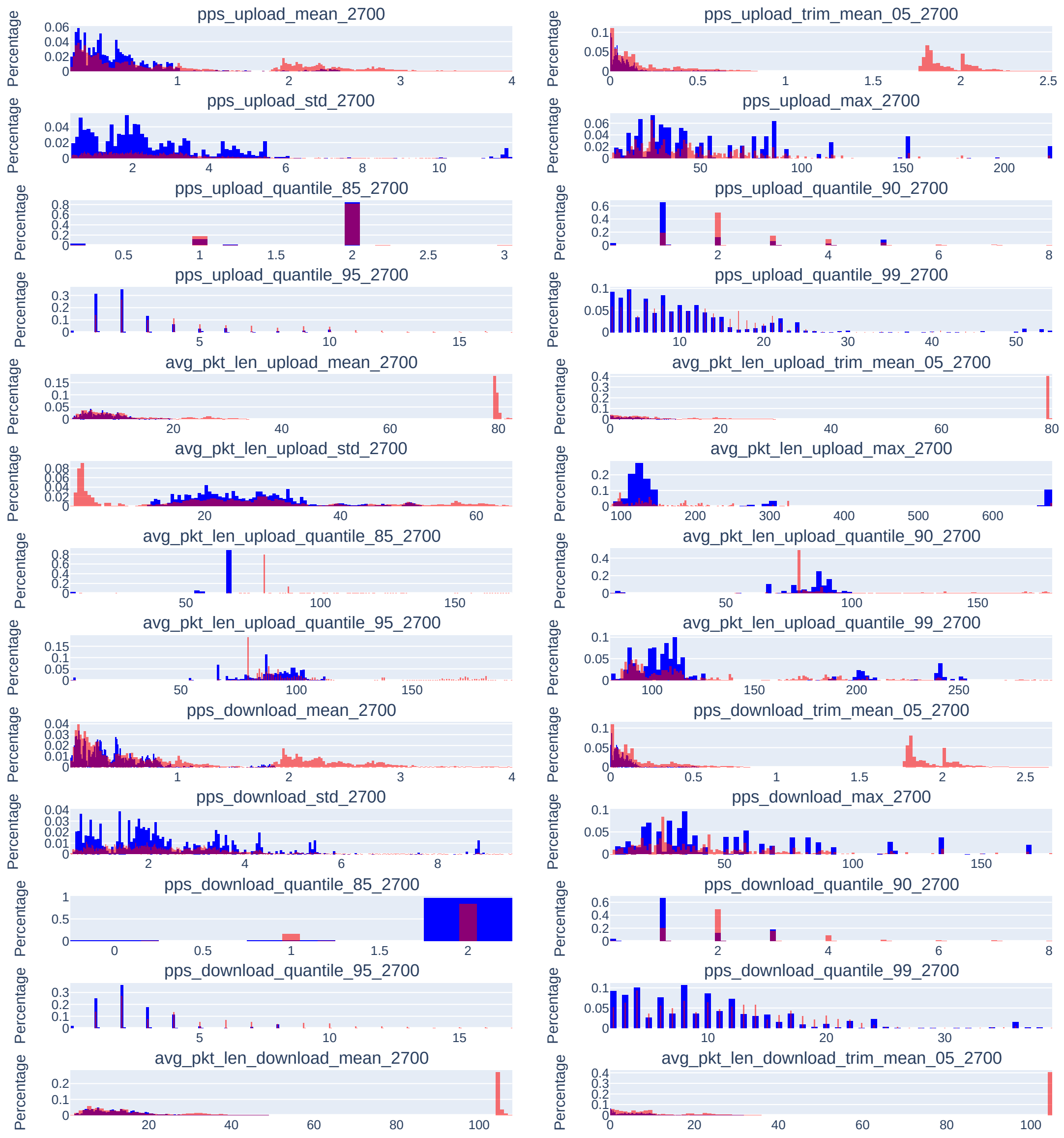


Figure 24: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 75 minutes - Part 1

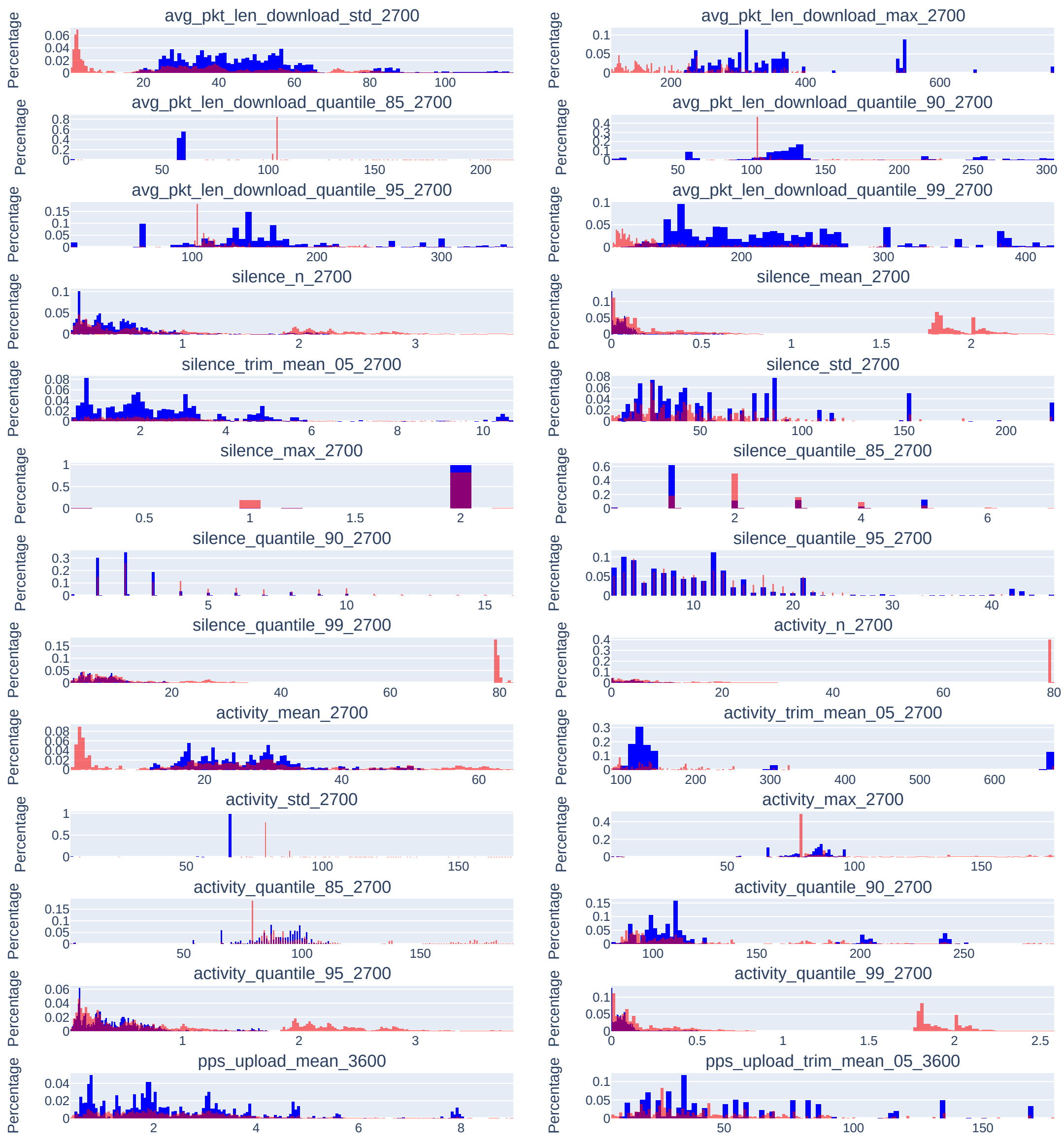


Figure 25: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 75 minutes - Part 2

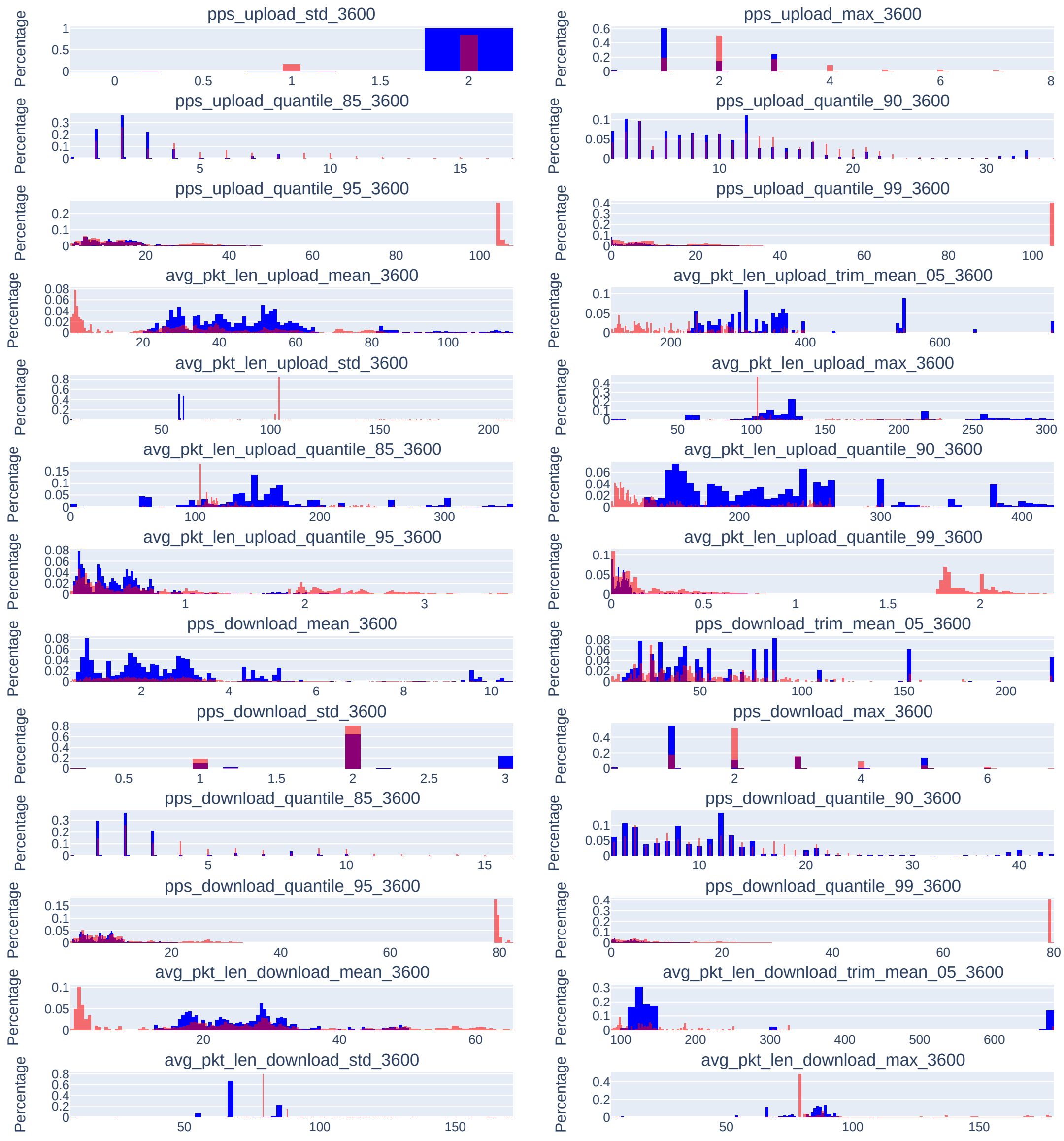


Figure 26: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 75 minutes - Part 3

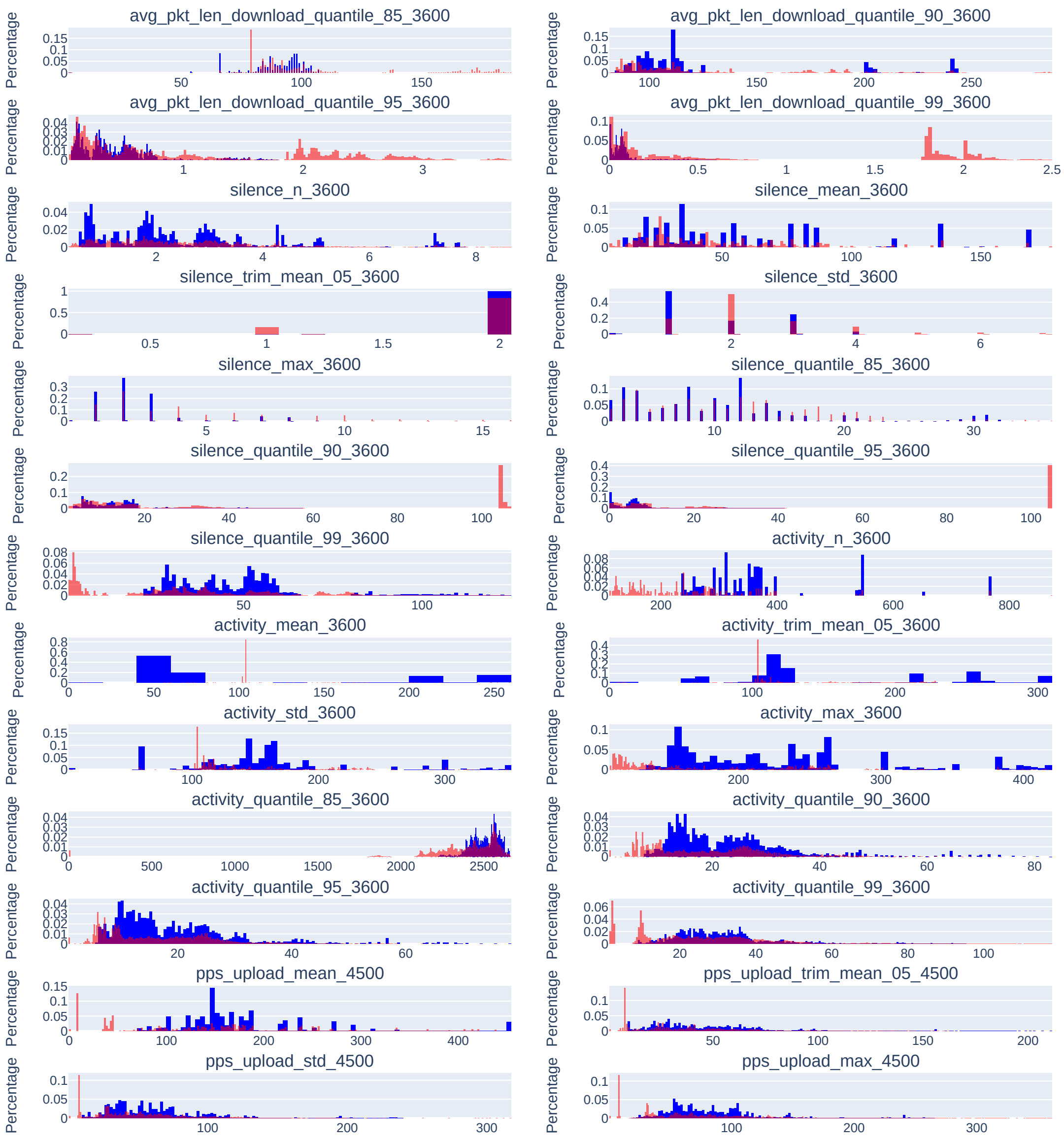


Figure 27: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 75 minutes - Part 4

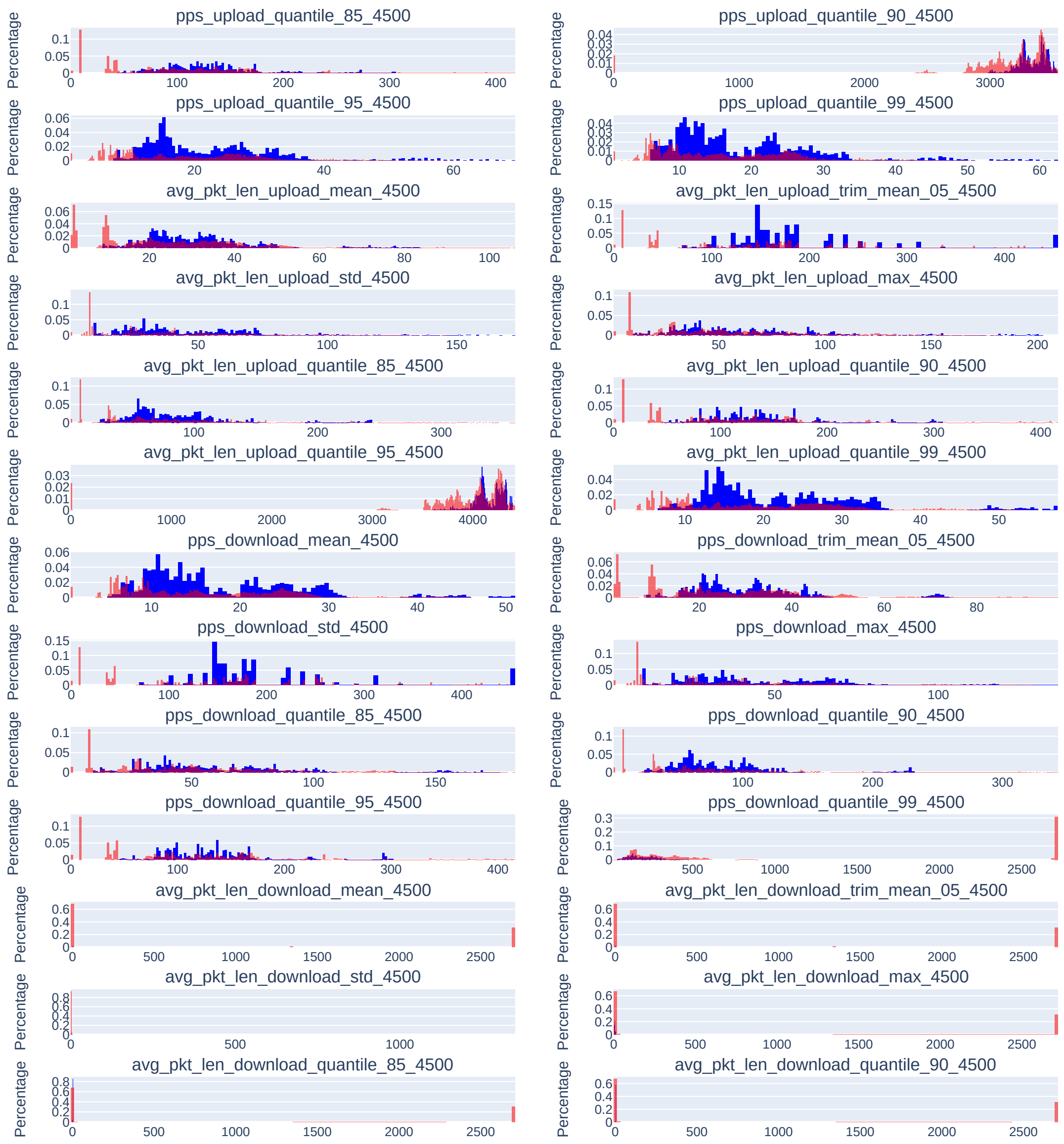


Figure 28: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 75 minutes - Part 5

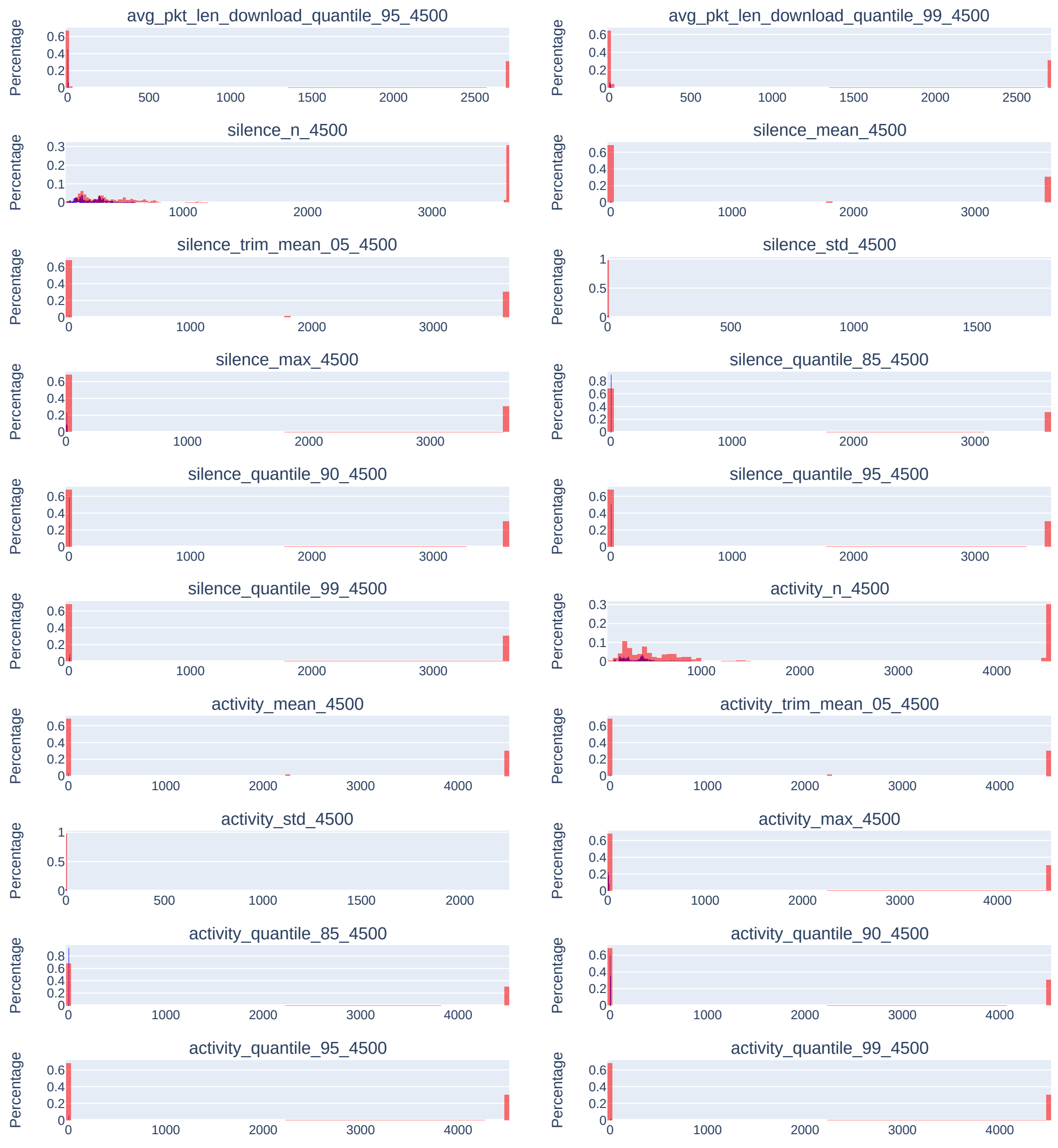


Figure 29: Comparison between licit (in blue) and illicit (in red) data points with and without *keep-alive* messages for the Dataset with a sliding window of 75 minutes - Part 6