Detecting and Mitigating the New Generation of Scraping Bots

Dissertation

submitted to

Sorbonne Université

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Author:

Elisa Chiapponi

Publicly defended on 07/11/23 before a committee composed of:

Examiner/Reviewer	**Prof. Roberto di Pietro**	RC3, CEMSE, KAUST, SA
Examiner/Reviewer	**Prof. Benoît Donnet**	Université de Liège, BE
Examiner	**Prof. Davide Balzarotti**	EURECOM, FR
Examiner	**Dr. Leyla Bilge**	NortonLifeLock, FR
Examiner	**Dr. Olivier Thonnard**	Amadeus IT Group, FR
Examiner	**Prof. Xianghang Mi**	USTC, CN
Thesis Director	**Prof. Marc Dacier**	RC3, CEMSE, KAUST, SA

The research has been conducted in the Digital Security Department at EURECOM (Sophia Antipolis, FR) in collaboration with Amadeus IT Group (FR) from April 2020 to August 2023.

Abstract

Every day an invisible war for data takes place between e-commerce websites and web scrapers. E-commerce websites own the data at the heart of the conflict and would like to provide it only to genuine users. Web scrapers aim to have illimited and continuous access to the above-mentioned data to capitalize on it. To achieve this goal, scrapers send large amounts of requests to e-commerce websites, causing them financial problems. This led the security industry to engage in an arms race against scrapers to create better systems to detect and mitigate their requests.

At present, the battle continues, but scrapers appear to have the upper hand, thanks to the usage of Residential IP Proxies (RESIPs). In this thesis, we aim to shift the balance by introducing novel detection and mitigation techniques that overcome the limitations of current state-of-the-art methods. We present two new detection techniques based on network measurements that identify scraping requests proxied through RESIPs. We propose a deceptive mitigation technique that lures scrapers into believing they have obtained their target data while they receive modified information. Thanks to an ongoing collaboration with Amadeus IT Group, we validate our results on real-world operational data.

Being aware that scrapers will not stop looking for new ways to avoid detection and mitigation, this thesis provides additional contributions that can help in building the next defensive weapons for fighting scrapers. We propose a comprehensive characterization of RESIPs, the strongest weapon currently at the disposal of scrapers. Moreover, we investigate the possibility of acquiring threat intelligence on the scrapers by geolocating them when they send requests through a RESIP. Finally, we provide future research directions to build upon the contributions of this thesis as well as ideas to exploit these contributions in other security use cases.

Résumé

Chaque jour, une guerre invisible pour les données se déroule entre les sites de commerce électronique et les acteurs qui,en siphonnent les données, sont appelés “scrapers” . Les sites de commerce électronique détiennent les données au cœur du conflit et souhaitent les fournir uniquement aux utilisateurs légitimes. Les scrapers veulent un accès illimité et continu aux données susmentionnées pour en tirer profit. Pour atteindre cet objectif, les scrapers envoient de grandes quantités de requêtes aux sites de commerce électronique, ce qui leur cause des problèmes financiers. Cela a conduit l'industrie de la sécurité à s'engager dans une course aux armements contre les scrapers afin de créer de meilleurs systèmes pour détecter et contrer leurs activités.

À l'heure actuelle, la bataille se poursuit, mais les scrapers semblent avoir le dessus, grâce à leur utilisation de Proxies IP Résidentiels (RESIPs). Dans cette thèse, nous visons à rééquilibrer la balance des forces en introduisant de nouvelles techniques de détection et d'atténuation qui surmontent les limitations des méthodes actuelles. Nous présentons deux nouvelles techniques de détection basées sur des mesures de réseau qui identifient les requêtes émanant de scrapers cachés derrière les infrastructures RESIP. Nous proposons une technique inspirée des “pots de miel” qui piège les scrapers en leur faisant croire qu'ils ont obtenu les données visées tandis qu'ils reçoivent des informations modifiées. À travers un partenariat en cours avec Amadeus IT Group, nous validons nos résultats en utilisant des données opérationnelles.

Conscients que les scrapers ne cesseront pas de chercher de nouvelles façons d'éviter la détection et l'atténuation, nous offrons des contributions qui peuvent aider à élaborer les prochaines armes défensives pour lutter contre les scrapers. Nous proposons une caractérisation complète des RESIPs, la plus puissante arme actuellement à la disposition des scrapers. De plus, nous examinons la possibilité d'acquérir des renseignements sur les menaces liées aux scrapers en les géolocalisant lorsqu'ils envoient des demandes via un RESIP. Enfin, nous proposons des orientations de recherche futures pour s'appuyer sur les contributions de cette thèse ainsi que des idées pour exploiter ces contributions dans d'autres cas d'utilisation de la sécurité.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's world, the advancement of technology and the Internet have led to the digitization of numerous tasks that were traditionally performed physically. Among these tasks, we find going shopping and performing bookings. Nowadays, indeed, everyone can buy all kinds of goods, from shoes to books, from events to airplane tickets with a simple click on their devices. There is thus a proliferation of e-commerce websites[1] listing goods available to purchase and their corresponding prices. While these websites are appealing to genuine users interested in buying goods, the information available there is also attractive to another kind of actors, called scrapers.

Scrapers are individuals or entities that have an interest in continuously extracting information from a website. This activity is called web scraping and is performed to gain a business advantage.

An example of scrapers is those actors that provide the best price on the Internet for specific goods (e.g. sneaker bots [1]). After scraping the prices of the goods on different websites, they present the best ones on their web pages. They then generate revenues thanks to user interactions within the page, e.g. advertisement, or obtaining a commission when users proceed to purchase the goods using the links provided by them. Another example consists of e-commerce websites that want to attract customers by proposing cheaper prices with respect to their competitors. To achieve this goal, these platforms need to know the original prices on the websites of the competitors and they commonly obtain this information through scraping.

In all scenarios, scrapers need to obtain accurate and up-to-date information to effectively gain a business advantage. Consequently, web scraping is carried out with high frequency and generates large amounts of requests.

[1] We define e-commerce websites as those platforms in which goods owners show their products and customers can perform online purchases. We aggregate together different industries (e.g. retail, travel, tickets) in the concept of e-commerce.

To produce these requests, scrapers usually leverage networks of bots. Bot, short for robot, indicates a software application that performs automated tasks. Bad bots, those that perform malicious activities, have been a plague of the Internet since the 2000s with the early Distributed Denial of Service attacks against Yahoo!, Amazon.com, CNN.com, and other major websites [2]. Since then, bots have continuously evolved from relatively rudimentary pieces of software to very sophisticated components, performing various kinds of attacks and making it harder and harder to distinguish their requests from the ones of genuine users [3]–[6].

The substantial amount of requests initiated by scraping bots carries significant business implications for e-commerce platforms. These websites need to answer scraping requests, incurring operational costs but having a low probability of obtaining any revenue from them. Furthermore, scraping can result in a loss of customers because of the induced degradation of the quality of service. Finally, the amount of requests generated by scrapers can cause Distributed Denial of Service on e-commerce platforms.

Hence, an invisible war takes place every day between the e-commerce websites, which would like to stop scrapers from accessing their data, and scrapers, which, naturally, do not want this to happen. Over the years, the security industry has engaged in an endless technological arms race with scrapers to protect e-commerce websites. This has led to a continuous production of detection and mitigation techniques followed by evasion ones.

As of today, the battle is still ongoing, but scrapers are on the winning side. Current detection is performed on parameters that scrapers can easily modify. Regarding mitigation, scrapers can infer that they have been detected due to the side effects of the current techniques used for this task. In this scenario, scrapers simply identify the parameters (e.g. user agent, TLS fingerprint) used to detect them and they change their value to avoid the mitigation. Then, they continue their activity until they are detected again. At that point, they change again the value of the identifying parameters and the process repeats itself.

Moreover, nowadays scrapers have at their disposal a new weapon, Residential IP Proxies. These proxies enable scrapers to send out requests from residential devices shared with genuine users. This significantly complicates the process of distinguishing between genuine and scraping requests.

On the security industry side, there is a need for new methods to detect and mitigate scraping connections that enable a better defense of scraped websites.

The above-mentioned information tells us that there exists a mutual relationship between the detection and mitigation of scraping bots. Moreover, it is illogical to prioritize the enhancement of one task without simultaneously finding a viable solution for the other. Successfully identifying scraping requests is futile if an efficient mitigation technique is absent. Possessing a robust mitigation technique would offer little assistance if the accurate identification of scraping requests remains unsolved.

For these reasons, in this thesis, we contribute to helping the security industry win the war against scrapers by investigating and finding new solutions for both scraping detection and mitigation techniques. Furthermore, we are well aware that the ever-evolving and dynamic nature of scraping bots poses challenges in achieving a definitive win against them. It is necessary to constantly refine our defense methodologies and study our adversaries to adapt to their evolution. For this reason, we provide a characterization of Residential IP Proxies, the latest technology leveraged by scrapers, and early results in geolocating the area of the world from where scrapers exploiting them originate their requests. These contributions can help us build the next defensive weapons for fighting scrapers.

1.1 Problem Statement and Research Questions

Thanks to an in-depth study of the scraping detection and mitigation state-of-the-art, as well as real data obtained from a company actively fighting real-world scrapers targeting their domains (Chapter 2), we had the opportunity to have a complete view of the issues that the security industry is facing nowadays in this scenario. We identified two main problems, detailed as follows:

- **Problem (PR1)** Current identification techniques for scraping requests rely on elements that scrapers can conveniently alter (e.g. browser settings). This allows scrapers to swiftly adjust these parameters and evade detection once they realize they've been identified. As a result, the process of detecting scraping activities becomes a repetitive and time-consuming task. Analysts and dedicated applications tasked with detecting scraping requests must repeatedly identify new parameters around which they can group the requests associated with each scraping campaign. Moreover, while there is an increasing usage of Residential IP Proxies, current detection techniques cannot distinguish between requests sent directly from a residential device or proxy through it. This increases the probability of having false positives when differentiating scraping and genuine traffic and forces the defender to be more conservative in the detection process.

- **Problem (PR2)** Scrapers can infer their detection from the side effects of the current mitigation techniques. This makes it easy for them to understand that they have been detected and act on it.

From the identified problems, we can derive two corresponding research questions:

- **Research Question 1 (RQ1)** How can we identify scraping requests without relying on easily changeable parameters even when they originate from residential IP addresses?

- **Research Question 2 (RQ2)** How can we mitigate scraping traffic without letting the scrapers know that they have been identified?

In this thesis, we aim to answer these questions. In the first part of the thesis (Part I) we address RQ1, showing that it is possible to differentiate requests sent directly by a device from the ones proxied through it by exploiting differences at the transport layer of the two types of connections. Moreover, while technically feasible, evading these detection techniques would require major changes in the infrastructure leveraged by scrapers.

In the second part of the thesis (Part II), we aim to address RQ2 thanks to a new deceptive mitigation technique that lures scrapers into believing they have reached their target while serving them modified but plausible information. In this way, scrapers receive no information about being detected and, at the same time, we poison their data collection process.

Furthermore, we recognize the ever-changing nature of scraping bots, which makes it challenging to achieve a definitive victory against them. Therefore, in the third part of the thesis (Part III), we conduct a comprehensive analysis of the Residential IP Proxies and scrapers ecosystem. This analysis aims to provide insights and assist the security community in developing new strategies to detect and counter evolving tactics of scrapers.

In the next section, we describe the different contributions of each part of this thesis as well as the detailed outline of the manuscript.

1.2 Main Contributions and Thesis Outline

The manuscript starts with an introductory part, which comprises this chapter and the following one (Chapter 2). In this second chapter, we describe the background knowledge necessary to understand the main contributions of the thesis. Moreover, we position our contributions with respect to the current state-of-the-art.

Afterward, the manuscript continues with three technical parts. The first part (Part I) investigates new detection techniques for scrapers leveraging Residential IP Proxies. The second one(Part II) studies a new way to mitigate scraping traffic without providing any indication of detection. The third part (Part III) studies the RESIP and scraping ecosystem.

We conclude the manuscript in Chapter 9, where we provide future research directions to build upon the contributions of this thesis as well as ideas to exploit these contributions in other security use cases.

In the following, we present the content available in each technical part and corresponding chapters, underlying the main contributions of each part.

1.2.1 Part I: Server-side Detection of Residential IP Proxies Connections

Chapter 3 explains the ideas behind RTT_DETECTION and MRP_DETECTION, two novel server-side detection techniques based on network measurements for scraping connections proxied through Residential IP Proxies (RESIPs).

RTT_DETECTION leverages the fact that RESIP providers break the TCP session between client and server but they maintain the end-to-end encryption between the two parties. This is measurable at the server side through the comparison of the Round Trip Times at the TCP and TLS layer. MRP_DETECTION exploits a machine retransmission protocol used by specific RESIP providers when the contacted server does not acknowledge ACK packets.

In Chapter 4, we provide experimental results demonstrating the feasibility, validity, and practicality of our first detection technique (RTT_DETECTION). We present our 4 months-long measurement campaign performed with machines scattered all over the world and its successful results. Moreover, we study when specific client environments could result in false positives and propose solutions to reduce these false positive cases.

In Chapter 5, we present our 88 days measurement campaign to validate our second detection technique (MRP_DETECTION). Moreover, we disclose the peculiar common behavior that all studied RESIP providers use when contacting a non-acknowledging server. It consists of leveraging multiple exit points, located in different areas of the world, to possibly overcome geographic restrictions or connectivity problems.

1.2.2 Part II: Web Scraping Mitigation Through Deception

In Chapter 6, we present a new deceptive mitigation solution that consists in redirecting the scraping requests to a web application honeypot and its implementation in a real-world scraping scenario. In the honeypot, scrapers receive pages with the same structure as the original website but with modified content. In this way, scrapers cannot understand that their requests have been mitigated and, at the same time, they do not access the real content of the website. Moreover, the honeypot enables us to poison the information they scrape and which they monetize, causing financial damage on their side. Furthermore, with this platform we can study the semantics of the payloads and the distribution of the IP addresses of specific bots, providing new insights into their ecosystem.

1.2.3 Part III: Residential IP Proxies and Scrapers Ecosystem Analysis

In Chapter 7, thanks to the accumulated data regarding RESIPs in the previous chapters of the thesis, we provide novel insights about the inner working of RESIPs in terms of geographic distribution, types, management and amount of their machines. We show the similarities and differences among providers and we suggest a possible new IP-based approach for a RESIP detection technique. Furthermore, we present a mathematical model that does not confirm the claims of RESIP providers about the volume of IPs[2] available as their exit points.

Chapter 8 investigates the possibility of geolocating scrapers sending requests hidden behind RESIPs . It discusses the multiple challenges of the problem at hand and proposes *RTTlocator*, a geolocation algorithm that exploits the Round Trip Time differences in RESIP connections and multiple packet speeds to find the scraper location(s) behind a set of requests. Early results suggest that geolocation is possible when the parameters are optimized for a specific set of requests.

1.2.4 Publications

The work explained in the different chapters resulted in a number of publications. We present here a high-level view of which content corresponds to each

[2] In this thesis, we use the terms "IP " and "IP address" interchangeably to refer to an Internet Protocol address.

publication, while we refer to the introduction of each chapter for a detailed explanation.

The industrial analyses presented in Chapter 2 were published in [C1]. Two patents were filed to protect the ideas presented in Chapter 3 ([P1], [P2]). Chapter 4 is based on [C2]–[C4]. The results in Chapter 5 and Chapter 7 were published in [C5]. Chapter 6 content was published in [C6], [C7], [J1]. Chapter 8 initial idea was introduced in [C3] and preliminary work was conducted in [C8][3]. The rest of the chapter is based on ongoing work. The full list of publications is provided in *Author's Publications.*

[3] In this work, the author of this thesis supervised the first author work and directly contributed to the tasks of study concept, data collection and interpretation of results, while the study design and implementation were performed by the other authors.

Chapter 2

Background and Related Works

2.1 Introduction

The goal of this thesis is to find new ways to detect and mitigate scraping traffic. In this chapter, we provide background information about web scraping and its usage against e-commerce websites (Section 2.2.1). Moreover, we discuss the state-of-the-art techniques for identifying and stopping scraping requests (Section 2.2.2). Thanks to a collaboration with Amadeus IT Group, we present data and statistics about web scraping campaigns targeting real-world domains and the insight we obtained from them (Section 2.2.3).

In Section 2.3, we focus in particular on one of the technologies most used by advanced scrapers nowadays, Residential IP Proxies. In Section 2.4 and Section 2.5, we offer background information about the technologies that we leverage to reach our goal, honeypots and geolocation. For both of them, we provide information about the previous works on that topic and we position ours with respect to them.

The industrial analyses presented in Section 2.2.3 were published in [C1].

2.2 Web Scraping

2.2.1 Web Scraping and E-commerce Websites

Web scraping consists of the periodical or continuous retrieval of accessible data contained in web pages to use elsewhere [7]. Scrapers usually conduct this activity behind a network of bots that, under their control, perform the actual action of reaching the website and harvesting the content.

On the security industry side, the Open Worldwide Application Security Project (OWASP) recognizes web scraping as an automated threat that produces “one or more undesirable effects on a web application” [7].

However, web scraping is legally categorized as a grey area. Indeed, the generic action of collecting publicly available data by itself is not illegal and there are no direct laws addressing it. However, if the website owner explicitly prohibits web scraping activity in their terms of service, this results in legal repercussions (e.g. LinkedIn vs HiQ [8]). Moreover, the illegal side of web scraping is mostly linked with the afterward usage of the scraped information to gain business advantage. Web scraping is performed for different reasons, among which are monetization of data aggregation, content reselling, statistics modifications and competitors’ monitoring. The scraping actor can use directly the information gathered through this practice or sell it for an income.

Scrapers target a wide range of markets [9] and in particular e-commerce websites where they retrieve the price and availability of the goods on sale. Moreover, scrapers change their activities according to the global situation [10]. With the COVID-19 pandemic reducing incomes from the traditional sectors, e.g. tickets for events, scrapers modified their actions by targeting websites selling medical devices, such as masks. Furthermore, bots profit from peaks of traffic due to holidays, e.g. the Lunar New Year [11], to increase their traffic without being noticed.

E-commerce websites provide prices of the goods they sell. Displaying this information comes with costs, such as maintaining the servers in which the information resides, retrieving the data, performing calculations, and rendering the final page to the client. Usually, the business model of these websites takes into account that only a few items among the displayed ones will end up being bought. Hence, the revenue for the sale of an item covers the costs associated with the maintenance of the whole infrastructure. Naturally, scraping affects this business model. According to DataDome, an anti-bot company [12], the loss could be up to 10% of the revenue of a website.

The income reduction can be direct and/or indirect. The direct one is caused by the dramatic rise in the number of requests which is not followed by an increase in the number of purchases. Moreover, scraping interferes with the metrics used to evaluate e-commerce business.

The indirect loss of revenue is caused by scraping bots representing large portions of the traffic towards a website and causing congestion and slow connections. This situation can reduce the number of legitimate users reaching the website and/or downgrade their user experience. In both cases, the company could lose a potential client. Finally, when the amount of traffic produced by these bots goes beyond the capacity of the scraped website, it results in a Distributed Denial of Service that makes the website completely unavailable.

For all these reasons, e-commerce websites aim to detect and mitigate those requests coming from scrapers while answering correctly the ones originated by genuine users. In the following section, we will provide an overview of all the techniques used to reach this goal.

2.2.2 Web Scraping Detection and Mitigation

Initially, scraping detection was performed based on IP blocklists with in-house solutions. Then, scrapers started to leverage larger networks of bots to send out their requests. This made the usage of IP blocklists ineffective [10]. Moreover, scrapers techniques evolved from simple scripts to browser emulation frameworks (e.g. Scrapy [13], Phantom JavaScript [14]) and complete automated browsers (e.g. Selenium [15]) able to run JavaScript, store cookies and imitate a real human interaction (e.g. mouse movement) [16].

Specialized anti-bot companies emerged to counter scraping activities. They position their products in front of scraped websites and investigate each incoming request. While we refer to [17] for a thorough explanation of the features of the most used anti-bot solutions on the market, we can categorize anti-bot solutions in two main detection approaches, the *knowledge*- and *behavior*-based ones [18].

The first approach consists of recognizing scrapers from their so-called bot signatures, the fingerprints of their requests. The most common types of fingerprints are browser and TLS fingerprints. Browser fingerprints consist of a set of information about hardware, operating system, and browser configuration of the used device [19]. This information is obtained thanks to application layer tests. Examples of browser fingerprint approaches can be found in [20]–[24]. We obtain TLS fingerprints studying the transport and network layers and capitalizing on the differences between browsers and environments at this layers [25]. Examples of such TLS fingerprinting approaches are [26], [27].

In the *knowledge*-based detection approach, custom rules associate mitigation techniques to requests matching a bot signature. The main problem with this approach is keeping these rules up to date. Current state-of-the-art mitigation techniques, as discussed in the following paragraphs, have the drawback of revealing that matching requests have been identified as coming from a bot. Thus, scrapers can understand they have been detected and react by simply rotating some parameters to modify their bot signature. This nullifies the previously imposed rule. Analysts need to identify a new signature and associate it with a mitigation technique. However, once again, scrapers can change their parameters and this results in a never-ending arms race.

Moreover, sophisticated scrapers rotate regularly their parameters before receiving any mitigation. In this way, even the process of constructing the initial bot signature becomes more challenging.

In a *behavior*-based approach, machine learning is privileged and the detection is performed by detecting outliers. Requests in which http headers and/or payloads differ significantly from the ones issued by known human beings are considered to be coming from bots and are answered with a countermeasure ([20], [21]). However, the usage of automated browsers running JavaScript and imitating real human interactions makes detecting outliers a much more difficult task.

In terms of mitigation, the most straightforward method involves blocking identified scraping requests. Another widely used strategy, which can help mitigate the impact of potential false positives in detection, is serving captchas [28]. These tests necessitate a human interaction to be solved. However, the effectiveness of captchas decreased with the emergence of captcha farms [29]. Scrapers redirect these tests to such infrastructures, where real individuals are compensated to solve them. captchas farm workers are required to solve the tests within a timeframe comparable to that of regular users, preventing anti-bot systems from flagging them. Another evasion approach involves redirecting the captchas to unsuspecting users on other websites and letting them solve them [30]. Additionally, in recent years, there has been a surge in the popularity of captcha-solving websites (e.g. [31]). These platforms operate as virtual captcha farms, enabling individuals to earn money by solving captchas on these websites.

Lately, a common mitigation approach consists of wasting the scrapers' time, hoping to decrease their revenues. New solutions are crypto challenges that make bots waste cpu cycles in solving them [32]. Other techniques, such as tarpits [33], consist of slowing down the bot connections or giving them open connections with no responses.

Nonetheless, as anticipated when discussing scraping bot detection, these mitigation strategies inadvertently signal to scrapers that their activities have been identified. Indeed, when requests are blocked, met with a captcha challenge, or encounter unanticipated delays, the scraper becomes aware of the enforcement of a rule and can easily modify his bot signature to evade detection. In Chapter 6, we propose a new mitigation technique that, by leveraging a honeypot (Section 2.4), overcomes this limitation.

Recently, scrapers have begun employing Residential ip Proxies (resips), which add complexity to the detection process due to the possible increase in the false positives ratio. In Section 2.3, we provide detailed background information about these parties, but first, in the next section, we study the problem of web scraping in one of the largest players in the it travel industry and we show the impact of RESIPs in this scenario.

2.2.3 Web Scraping in Amadeus IT Group

Amadeus IT Group, later referred to as Amadeus, is a Global Distribution System (GDS) and one of the world's leading technology companies for the IT travel industry. In 2019, more than 646 million bookings were processed by Amadeus and 1.9 billion passengers were boarded thanks to its portfolio of IT solutions [34]. In the context of this thesis, we worked in close collaboration with their Security Operation Center (SOC).

Among the various products offered by Amadeus, there are solutions specifically built for airlines to let passengers make bookings on airline websites. These products share a common back-end, in which Amadeus calculates for each customer request the possible flight routes and their corresponding price. These fares are computed in real-time and based on a large number of parameters e.g. origin-destination, departure and arrival dates, travel classes, passenger types, etc, but also the availability of seats, the period of the year, and many other business rules. Thus, every generated response comes with a high computation cost.

Clearly, web scraping increases dramatically the costs for providers like Amadeus and its customers. Moreover, one of the Key Performance Indicators for an airline is the look-to-book ratio. This value is the ratio between the number of searches made by website visitors and the actual number of bookings made on the airline website. The high volume of traffic generated by scraping bots toward airlines' websites is degrading this important indicator. As highlighted by the anti-bot company Imperva [35], the travel industry is one of the most targeted sectors and the one in which the percentage of advanced scraping bots is predominant: 70.3% of the overall bot traffic.

Since 2015, Amadeus has used a *knowledge*-based Bot Detector from a third-party company to protect more than 80 airline companies, corresponding to more than 200 websites. This product detects bots thanks to fingerprinting. It defines the bot signatures, a series of parameters that identify requests coming from different IP addresses but associated with the same source scraper. Each bot signature can be linked to a countermeasure, producing a custom rule.

Every week more than 100 new custom rules are put in place by the Amadeus SOC to mitigate the scraping traffic and, every month, around 140 million requests trigger these rules. In a representative month (February 2022), on average, 8 intense bot investigations on specific airlines were running every week to mitigate the huge amount of bot requests. In the same month, considering all the domains protected by the anti-bot solution, that product adjudicated that 41% of the attempted connections to the Amadeus servers were issued by bots.

After observing the efforts of the Amadeus SOC team in fighting scraping, we have gained the following insights:

- *Blocked bots die*: As soon as a certain type of bot is blocked, the traffic associated with that bot disappears. In other words, the Bot Detector has no match anymore against that signature. This means two things about the bot operators. First, they continuously verify the stealthiness and efficacy of their bots. They do not waste their resources by sending requests that do not bear fruits. Second, they can modify their bots extremely rapidly (within minutes, or even seconds) to avoid the detection mechanisms put in place against them. We assume that they have at their disposal analysts and/or systems that monitor the bots 24/7 and act swiftly to bypass the custom rules as soon as the countermeasure is put in place.

- *Scrapers verify the harnessed information*: If the information provided to a bot is completely different from the real one, e.g. a different web page, the traffic associated with that bot disappears, almost immediately (within minutes). This means that the bot operators deduce from the feeding of incorrect information that their bots are now identified and mute them to become stealthy again.

- *The risk of having a false positive is high if scrapers use residential IPs*: When scrapers take advantage of residential IPs shared with real users, the risk of blocking a connection from a genuine customer is high and this prevents the SOC to put in place strict policies against those requests.

Scrapers gather residential IPs thanks to Residential IP Proxies (RESIP) providers and this practice has increased in the last years [36]. We have identified this problem in Amadeus as well. A growing fraction of their traffic flagged with bot signatures is coming from residential Internet Service Providers (ISPs). In representative 30 days (February 13th - March 15th 2022), Amadeus blocked more than 22M connections, considering all the protected airlines. Dividing these connections by ISP of origin, we see that a few providers account for the majority of the blocked connections. There are 43 ISPs of which more than 50K connections are blocked. This corresponded to 84% of the blocked traffic. Among these providers, we identify 13 organizations as mostly providing IP addresses belonging to residential use. Their traffic amounts to 12% of the blocked traffic of the period. This percentage could look small, but we have to keep in mind that Amadeus' goal, like other e-commerce websites, is to reduce to zero the probability of false positives. A decision to block a connection from a residential IP is only taken when the confidence is very high that it is a malicious (e.g. scraping) one. This data implies that the traffic received by scrapers through residential ISPs is a larger portion than this and thus shows how wide their usage is.

Moreover, we have personal experience of RESIP forwarding requests to Amadeus. During an investigation, a member of the SOC team found out that many bot requests were produced from IP addresses registered by one specific RESIP company. Most likely, there had been a problem in their setup and they were using their machines instead of residential ones to proxy the traffic.

The data in this section explains the practical limitations that analysts encounter in their day-to-day jobs. In particular, it shows evidence of the large usage of RESIPs to perform scraping and the challenges that this raises. In the next section, we focus on these parties, offering an overview of their characteristics and how scrapers leverage them.

2.3 Residential IP Proxies

2.3.1 Residential IP Proxies and Web Scraping

In the past, scrapers leveraged datacenter and compromised machines to perform their activities. Recently, this trend has changed and scrapers take more and more advantage of Residential IP Proxies (RESIP) providers. These companies offer access, for a fee, to a network of residential and mobile devices, e.g. phones and laptops, shared with real users. Scrapers can use these devices as exit points for their requests. In this way, they send requests showing residential[4] IP addresses.

As displayed in a blog post of DataDome [36], a company dedicated to detecting bots, RESIP IPs counted for almost 30% of the malicious bot traffic at the end of the year 2019. In [25], Li et al. show that more than half of the bot IP addresses that they collected with their honeypots belong to residential networks. Imperva, an anti-bot company, explains that in 2021 more than half of the bot traffic (54.9%) originated from residential and mobile ISPs [35]. This data shows how RESIP connections became an important party in the bot ecosystem.

Scrapers take advantage of RESIP for multiple reasons. First, using RESIP networks, they do not need to privately build or maintain a distributed

[4] In the rest of this work, we call residential IPs those belonging to residential and mobile ISPs.

infrastructure to send requests from multiple IPs. This reduces the efforts on their side. Moreover, they share an infrastructure, that they do not own, with other parties. This lowers the possibility of directly tracing back a, possibly illegal, activity to them.

Furthermore, real users share the usage of their devices with RESIP providers. At the same time, they perform legitimate activities on the Internet with them. Scrapers leverage the good reputation built with these activities to evade detection. Indeed, bot detectors privilege reducing false positives over false negatives to not lose potential customers of the protected websites. When a request shows a residential IP, bot detectors need to identify if the source device sends the query directly or proxies it out. This is a difficult task because both queries are indistinguishable at the application layer, where the majority of detection techniques act. Thus, in most cases, these requests do not receive a countermeasure to prevent false positives. This enables scrapers to perform their activities without mitigations.

Finally, some RESIP companies offer automated services to show human activity (e.g. automated CAPTCHA [28] solving and JavaScript rendering) and to reduce the correlation among requests in the same campaign (e.g. rotation among different fingerprints) [37]. This helps in overcoming detection and enables not proficient developers to easily conduct their scraping campaigns.

RESIP providers advertise to have at their disposal tens of millions of residential IPs. Mi et al. experimentally confirmed this, showing that 95.22% of the RESIP IPs they collected (6M+) was indeed residential [38]. Contrary to datacenter IPs, residential ones are dynamically assigned by ISPs. Thus, a user proxying out a request through the same device in different moments, especially a mobile one, could end up sending requests with different IPs. Moreover, different devices part of a RESIP network can obtain the same IP address. This happens if a device changes IP and a second one takes the previous IP or if the two devices are behind the same Network Addresses Translator (NAT) device. For these reasons, there is no 1-1 correspondence between the number of devices and the number of IPs involved in a RESIP network.

RESIP companies claim to legally have access to the devices they use as proxies. However, their device recruitment processes raise concerns. Some RESIP providers build their infrastructures taking advantage of mobile Software Development Kits (SDKs). Developers can include these SDKs in their applications and obtain a fee per installed app. Device owners voluntarily download these applications and implicitly give consent to be part of the network [39]. In theory, RESIP providers require developers to inform end users about the RESIP SDK in the terms of service. However, there is no proof that they strictly implement or monitor this policy [40]. On the other hand, there is evidence that some providers lure device owners. Frappier et al. [41] explain that a provider makes end users install software that looks legitimate in exchange for a free VPN service and uses their devices as proxies without their informed consent.

Furthermore, infected devices, such as IoT ones, represent a consistent percentage of RESIP devices [38]. RESIP providers also use browser extensions as a vehicle of proxy activity and a recent approach consists of building mobile proxy networks using dedicated hardware. In this way, RESIP providers create large clusters of SIM cards and use them to route the traffic towards mobile networks [42]. In recent years, researchers have performed studies on the RESIP recruitment process and have proposed different approaches to detect if a device is part of the networks [39], [40], [43].

Besides web scraping, the IPs of RESIP exit points have been associated with other malicious activities while part of these networks. RESIP networks provide anonymity when doing automatic ad clicks on commercial platforms, generating new accounts, performing credential stuffing attacks as well as producing social media spam [44]. RESIP IPs have been involved in malicious website hosting [38] and IoT botnet campaigns [45]. Moreover, some RESIP providers use “potentially unwanted program” to relay traffic and their IPs has served as fast flux proxies [38]. RESIP IPs have been involved in cryptojacking activities [45] and cyber criminals exploited this type of IPs to masquerade their identity in political contexts [46]. Finally, known VPN providers leverage RESIPs to avoid geolocation blocking of streaming services [47].

This data clearly reveals the malicious usage of RESIP networks and their IPs. It shows why it is important to have a complete understanding of these new actors and why we need new methods to identify when they are used in communications. In the rest of the section, we will provide an overview of the works performed so far in this context. Moreover, we position the contributions we provide in this work with respect to the state-of-the-art.

2.3.2 Past Works in RESIP Analysis and Detection

RESIP Analysis

In 2019, Mi et al. proposed the first comprehensive study of RESIP services [38]. They created an infiltration framework and they used it in 2017 to collect 6M+ unique IPs from 5 RESIP providers. To obtain one of the RESIP datasets studied in this work (RTT_DS), we use an infrastructure similar to theirs. With it, we obtain a much larger dataset than the one collected in their work (13M+ unique IPs).

Mi et al. disclosed internal features of RESIP services, but they focus more on investigating the involvement in malicious activities of RESIP IPs and studying if these addresses are residential, as advertised by the providers. Differently from them, we examine with much more depth geographic distribution, types, management and amount of RESIP IPs in RTT_DS. This enables us to obtain original findings about the internal functioning of RESIPs.

After this initial work, different studies about RESIP services have been proposed. In 2020, Choi et al. [48] used the dataset of Mi et al. to compare RESIP GATEWAYs and open proxies. The authors study the locality distribution of these parties and check the reputations of the collected IPs. In the same year, Hanzawa et al. [49] used the same dataset to better characterize RESIP GATEWAYs in Japan and identify connected malicious activities to those GATEWAYs. Differently from those two past works, we consider completely new datasets, our work does not focus only on one specific region of the world and our analyses concern repetitions and assignation patterns of our IPs.

In 2022, Yang et al. [45] proposed a characterization of the RESIP ecosystem in China. They investigated how many different RESIP services exist and how they work. To study the back connected RESIP providers, they use an infrastructure similar to the one in [38], as we do. With it, they have collected a smaller dataset with respect to ours (9M+ IP addresses). After a short analysis of the ecosystem, whose techniques we share to find one out of our ten findings about the RESIP ecosystem (Chapter 7), they focus on understanding the security risks associated with RESIP IPs. In comparison, we propose a worldwide approach and collection. Furthermore, we perform a much deeper analysis of the ecosystem.

Finally, in this thesis, we propose a novel dataset (UNACKED_DS) that has been collected with a brand-new setup with respect to all previous works. Thanks to this dataset, we disclose additional new insights about the algorithms RESIP services use.

RESIP Detection

In this thesis, we propose two new different approaches to identify RESIP connections, RTT_DETECTION and MRP_DETECTION.

In 2022, a blog post from the anti-bot company DataDome [50], proposed a ML based approach to identify RESIP connections. They claim that RESIP IPs exhibit a different behavior compared to residential IPs used only by humans, even when the IPs are shared by the two categories. Thanks to an infrastructure similar to the one in [38], they collect RESIP IPs. Every time they detect one of these IPs sending them requests, the ML model checks if the request exhibits RESIP behavior. This approach, different from ours based on network measurements, might prove to be effective. However, the author only proposes a high-level overview of the used behavioral features and this method most likely requires a large dataset of connections to be effective. On the contrary, our methods can systematically and deterministically detect a RESIP connection by only analyzing a single request. Furthermore, MRP_DETECTION is able to identify connections from specific RESIP providers.

Proxy Detection based on Round Trip Time

While RESIP detection is a recent topic, researchers have studied in depth the broader field of proxy detection. One of our detection methods, RTT_DETECTION, relies on the comparison of the TCP and TLS RTT, measured between packets that are exchanged within a TLS session. To the best of our knowledge, we are the first to implement this technique.

Using some form of RTT measurement for proxy detection was proposed before, though. For instance, Hoogstraaten [51] suggests that comparing the RTT at the application layer and the transport layer could potentially indicate the presence of a proxy. He advises calculating the application RTT by retrieving consecutive elements from the server (e.g. HTML page and associate image) using HTTP. As far as we know, this technique was not implemented and this approach is different from the one we propose. Indeed, this method requires changes in the application code and only works if some specific assumptions hold (no caching in the proxy, no parallel requests to retrieve both objects, etc.). Our technique, on the other hand, does not require any modification of the original server code because it leverages the exchanges that normally take place in a TLS connection.

In a blog post [52], the author suggests a proxy detection based only on the measurement of the RTT_{TLS} (ignoring the RTT_{TCP}) which could, possibly, work thanks to an implementation issue in chromium-based browsers. His technique takes advantage of JavaScript code running on the client side. The code queries 5 times both 127.0.0.1 and 0.0.0.0 with HTTPS. In the case of direct connections, the RTT of the two connections are comparable. When the connection is proxied, there should be a relevant difference between the two measurements. This technique is different from ours for various reasons. Our detection method does not require any code running on the client side, is independent of the client application or operating system, and is solely based on measurements obtained on the server side. No additional URL needs to be queried and the comparison of RTT values is performed between the TCP and TLS RTTs of a single connection.

Other works leverage similar approaches to that blog post. In [53], a RTT at the application layer is calculated by fetching an HTTP object that cannot be cached. A patent [54] performs the detection with the comparison of the RTTs obtained when fetching a cached and a non-cached object. Our approach, in contrast to the described ones, works completely on the server side, does not require fetching any object and uses the more stable TCP and TLS RTTs as opposed to the ones at the application layer.

In summary, previous works have considered using some form of RTT measurement to detect proxied connections. However, they all require either modification of the server code and/or some JavaScript to be executed on the client side. Our proposal, in contrast, is exclusively based on passive measurement made on the server side, which does not need to be modified in any way. Furthermore, contrary to our work, none of the previous works compares the TCP RTT to the TLS one.

2.4 Honeypots

In Chapter 6, we present our deceptive approach, based on the redirection to a web application honeypot, to mitigate scraping requests.

The idea of deceiving an attacker to win him over is certainly not new and deception has proven to provide valuable information to defenders in the past. When it comes to computer security, one of its first incarnations dates back to 1986 with the famous Cuckoo's Egg story by Clifford Stoll. It recounts the true story of Stoll's investigation into a hacking incident at Lawrence Berkeley National Laboratory. The author leverages a system under his control to study the actions of the hacker and unmask him. A few years later, in 1992, B. Cheswick told us about his "evening with Berferd" [55]. The author observed and analyzed the actions of the hacker to shed light on his methods and motivations. After that, W. Venema created TCPWrapper [56], a network access control system for Unix systems that enables administrators to control which clients are allowed or denied access to various network services. TCPWrapper made it possible for everyone to play with the idea of deceiving attackers.

In [57], Cohen formalizes the notion of deception in computers. He writes that it is about exploiting errors in the cognitive systems of attackers for advantage. Moreover, he reviews the early works on honeypots. Honeypots are security deception mechanisms that reproduce a part of or a whole real system. They lure attackers reaching them into believing they are on the real system they want to exploit. Honeypots are used to detect, mitigate and study attacker behaviors. Depending on the level of interaction an attacker can have with the system, we classify it as low/mid/high interaction honeypot [58].

In the latest years, honeypots, honeynets and honeytokens [59] have been

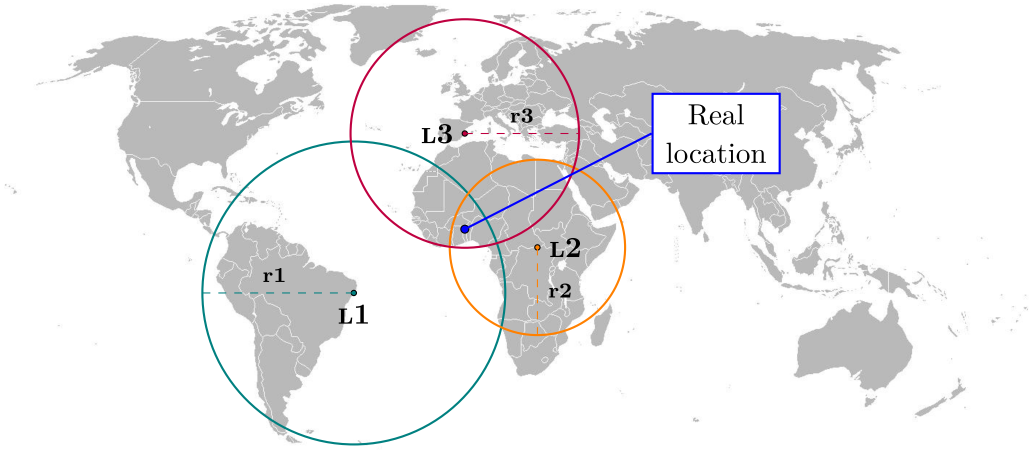


Figure 2.1: Multilateration using 3 landmarks.

widely used to perform deceptive activity against attackers. Some honeypots simply collect information about attackers to learn their modus operandi [60], [61] or derive actionable knowledge about them [62], possibly leading to attack attribution applications [63]. Others take an active role in slowing down the attackers. They are then usually called sticky honeypots, tarpits, or crawler traps [64], [65]. These honeypots aim to prolong attackers' engagement to make them lose time and gather more information about their methods at the same time.

In the past, Bait&Switch Honeypots [66] and the Intrusion Trap Systems [67] already proposed to redirect malicious traffic to a honeypot that mirrors the real site under protection, as we do in Chapter 6. The limitation of these approaches is the creation of simplistic versions of the real website as honeypots. The main differences, concerning the work we propose, lie in the complexity of the system we have to mimic, the sophistication of the attackers to lure, and our objective to provide plausible, yet inaccurate information.

2.5 Geolocation

2.5.1 Background

Geolocation consists of using technologies to obtain the geographical location of the initiator of an Internet activity [68]. We can divide geolocation into two categories, based on the techniques leveraged to achieve the goal: registration-based and measurement-based.

Registration-based techniques take advantage of one or more databases in which blocks of IPs are associated with a specific location. Once the IP of the machine to geolocate is known, we can query the database for that IP and obtain the location of the corresponding block. These databases can be both public (e.g. RIPE [69]) or proprietary (e.g. MaxMind [70]). The process behind building the proprietary databases is generally not public. However, from the available information, they should leverage whois services, DNS LOC records and Autonomous System numbers to perform geolocation [71]. Different works investigate the reliability of these databases and show their reliability at different granularity [72]–[76].

Measurement-based techniques, as the name suggests, leverage measurements from landmarks whose location is known, to the machine to geolocate. Thanks to this, they obtain information about the location of the machine. Generally, measurement-based geolocation is performed through multilateration. This is the "process of locating an object by accurately computing the time difference of arrival of a signal emitted from that object to three or more receivers" [77]. As displayed in Figure 2.1, given three landmarks (the receivers) L1-L3, we can build a circle centered in each landmark whose radius (r1-r3) refers to the distance between the landmark and the contacted object. The object to localize can be at any point on this circumference. The intersection of the circumferences created from each landmark reveals the position of the object. Ideally, the intersection is a point. However, due to measurement errors and delays, in practice, the intersection corresponds to an area of the world. Geolocation algorithms mainly rely on the RTT delay between machines on the Internet.

In this work, we leverage a registration-based geolocation dataset (MaxMind [70]) to retrieve the location of the RESIP gateways, whose IP addresses are collected in RTT_DS. In Section 4.2.4 and Section 8.5 we discuss the reliability of this database relative to our use cases.

Moreover, in Chapter 8, we try to solve a geolocation problem: geolocating the original client sending requests through a RESIP. In this context, the server does not have any information about the IP address of the original client but only has information about the RTTs between it and a set of GATEWAYs. Thus, only measurement-based techniques based on RTT can be used to perform geolocation.

However, the problem we want to solve is more complex than simply geolocating a machine that contacts directly a set of landmarks through RTT (Section 8.2). No previous measurement-based technique has addressed this scenario. Moreover, previous works needed more information and/or made assumptions that do not hold in our use case. In the following, we describe the state-of-the-art regarding measurement-based techniques based on RTT and we explain why each of them cannot be directly applied to our proposed scenario.

2.5.2 Past Works in RTT-based Geolocation

Different past works have used the RTT to geolocate the location of a machine. In [78]–[83], the RTT geolocation is restricted to specific regions of the world (e.g. connections in one continent) and need network topology information to decide which speed to apply in order to convert millisecond (RTT, time) into kilometers (distance). In our use case, connections are performed from locations all over the world and we do not have any information about the topology of the network. We geolocate a machine hidden behind a RESIP infrastructure, composed of an unknown number of machines. In our approach, we leverage the usage of a combination of speeds to try to model the different situations and topologies that could be found on the path of the packets.

Other works ([84], [85]) do not work on a global scale because of their need to have landmarks close to the solution. In our use case, the original solution is unknown. Furthermore, RESIP machines could be anywhere in the world and we do not have any control or information about their location. For this reason, these methods are not appropriate for our use case.

There are works that perform geolocation without the above-mentioned constraints ([86]–[88]). However, they apply their algorithm only in conditions of well-connected machines (Planet-Lab nodes, RIPE anchors) and perform multiple measurements from the same landmark to find the most accurate RTT value i.e. the least impacted by transient network latencies. In our setup, we do not have any control over the connectivity of the landmarks and we usually have only one connection from each of them. Indeed, we passively acquire the RTT values between the landmarks and the machine to geolocate and we cannot directly produce multiple measurements. Thus, we can not apply these methods.

In [C8], we created a RTT geolocation method for machines directly connecting with each other. The technique works for connections all over the world, without constraints about the connectivity or location of the landmarks and does not need multiple measurements from each of them. The results of the technique tell us that considering multiple speeds helps in refining the accuracy of the geolocation. This is a central point of our new algorithm. We leverage the work proposed in [C8] and modify it to better apply to our new setup. We use a novel algorithm as a preliminary phase to identify the sole connections that mostly contribute to the solution and to remove the ones that only add noise, due to the problem at hand. Moreover, we expand it to possibly identify multiple clients behind a RESIP.

2.6 Summary

In this chapter, we explored background information about web scraping and the techniques we use to detect and mitigate the corresponding traffic. Studying this scenario, we understood the limitations of the current approaches and the problems raised by the significant usage of Residential IP Proxies. Overcoming these limitations is the motivation for this thesis.

In the next chapter, we start addressing the problems we identified in the detection process by proposing two new identification techniques for scrapers exploiting Residential IP Proxies.

Part I

Server-side Detection of Residential IP Proxies Connections

Chapter 3

Identification of Scrapers Exploiting Residential IP Proxies

3.1 Introduction

Web scraping techniques keep evolving to enable scrapers to bypass any new detection and mitigation action against them. Recently, scrapers have started to take advantage of Residential IP Proxies (RESIP) to facilitate their operations.

In the previous chapter, we learned that RESIPs usage can be malicious and that we need new detection methods for this type of connection. Section 2.3 displayed how widely, and why, scrapers use RESIP. Moreover, it showed the RESIP involvement in other malicious activities (e.g. cryptojacking, credential stuffing attacks and many more).

Section 2.2.3 illustrated the growing impact of RESIP IPs in scraping one of the leading companies in the travel industry. It showed the challenges that analysts experience while fighting scrapers behind RESIPs. Current detection techniques have difficulties in discerning when a device sends directly a request from when it is used as a RESIP proxy. To reduce false positives, analysts tend to not block requests coming from residential IPs. In this way, scrapers exploiting RESIP services, get easy access to the contents they want to obtain.

In this chapter, we present the intuition behind two new server-side detection techniques, RTT_DETECTION and MRP_DETECTION. These techniques, based on network measurements, enable us to overcome the limitations of current scraping detection tools and they successfully identify RESIP connections. As discussed in Section 2.3, in 2022 DataDome proposed another method [50]

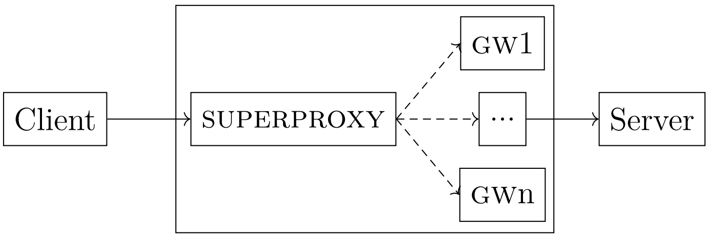


Figure 3.1: Common internal structure of RESIP providers.

to identify RESIP connections. This approach, based on machine learning, might prove to be effective. However, it needs to be trained with a likely large dataset of connections to have a good detection rate. Differently from it, our techniques can systematically and deterministically detect a RESIP connection by only analyzing a single request. Furthermore, we can exploit one of them to identify connections from specific RESIP providers.

Our two techniques leverage features of the RESIP infrastructure. Thus, we first provide an overview of the internal functioning of these parties (Section 3.2). We take as assumptions that the TCP sessions between different parties of a RESIP connection are not synchronized among them and that RESIP machines do not break the HTTPS end-to-end encryption between client and server. Section 3.3 shows that these assumptions hold after experimenting with four among the most used RESIP providers. Finally, Section 3.4 and Section 3.5 explain the rationales behind respectively RTT_DETECTION and MRP_DETECTION.

The ideas underlying our techniques are protected in two filed patents [P1], [P2].

3.2 Residential IP Proxies Infrastructure

RESIP services do not share details about their internal infrastructures. From the available information, we know they generally work in, so-called, back connect mode. Fig. 3.1 shows the schema of a back connected RESIP, as discussed in previous works [38], [89], [90]. In this setup, the client sends a request to the so-called SUPERPROXY. The SUPERPROXY forwards this request to one of the residential GATEWAYs (GW1,...,GWn).

In [38], the authors found that there is a series of backend servers intermediating between the SUPERPROXY and the GATEWAYs for specific providers. It is not clear if this is a common behavior among other providers. While this is not an important point for the outcome of the detection techniques we present hereafter, we take this into account in Section 8.2 when geolocating the original client behind the RESIP. Finally, the request arrives at the server with the IP of the GATEWAY as source IP and does not contain any application-level information that could indicate that it has been proxied.

RESIP clients can specify the location of the GATEWAY to use for their requests. Furthermore, they can decide if the same IP must be used for a series of requests or if, instead, a new one must be provided for each new request they want to send. Different RESIP providers implement various proxy protocols, but generally, they all support HTTP/HTTPS. In this case, the client contacts the SUPERPROXY with a HTTP CONNECT to establish the communication.

Studying RESIPs, we have reached some conclusions about their functioning. RESIP machines have to perform SSL tunneling to enable HTTPS connections. This means that they do not decrypt and re-encrypt the packets they receive. They simply act as a circuit gateway by forwarding packets back and forth between 2 (or more[5]) distinct TCP sessions while changing IP addresses. Moreover, these TCP sessions are not synchronized among them. In the next section, we experimentally confirm this description.

In [45], the authors discover that some RESIP providers offer a second operating mode called direct. In the case of direct RESIP, the client acquires the IPs of the GATEWAYs and contacts directly the one(s) chosen to proxy requests out. This type of RESIP looks popular mostly for Chinese providers but it does not appear to be widely used globally.

Considering both RESIP operation modes, we can conclude that RESIP infrastructures are merely instruments in the hands of parties exploiting them (e.g. scrapers). These parties do not own the network or the SUPERPROXY machines, they can just rent the provided service and use it through a well-defined API. They cannot access the internal parts of the infrastructure and change their functioning.

Similarly, RESIP providers take advantage of devices belonging to real people. RESIP providers do not own these devices and cannot access them directly. Thus, they cannot alter the hardware of the devices and they can only use their application-level features.

These constraints create a fixed environment in which scrapers send their requests. As explained in the next sections, we can exploit some of its features to create identification methods for connections traversing such an environment.

[5] Depending on the setup and the number of backend servers that are part of the infrastructure, the number of TCP sessions built by the RESIP can vary.

3.3 Confirmation of Working Assumptions

In the previous section, we have made two assumptions about the functioning of RESIP infrastructures:

1. RESIPs build distinct TCP sessions among the parties taking part in a connection and these TCP sessions are not synchronized among them

2. RESIP providers do not break the TLS session but just perform SSL tunneling

We experimentally confirmed this behavior using four of the most used RESIP providers (Bright Data[6] [91], Oxylabs [92], Proxyrack [93], and Smartproxy [94]).

To check their validity, we first consider a server that does not send any ACK packet. In this scenario, a client sending a SYN packet to this server does not receive any answer and can not complete the TCP three-way handshake. If the RESIPs simply forwarded the TCP packets and changed the source IP address, a client sending requests through them would never complete the handshake. The same outcome would take place if the RESIPs built distinct, yet, synchronized TCP sessions. In such case, the client would receive an ACK packet from a RESIP machine only if another machine in the RESIP infrastructure had previously received an ACK packet from the server. Hence, since our server does not send any ACK packet, the client should not be able to complete the TCP handshake.

We tested connections passing through the four RESIP providers in the scenario just discussed. From one client, we tried, with each of the four providers, to connect to the server that does not send ACK packets. On the client, the three-way handshake was always completed successfully. This means that the SUPERPROXY sent the ACK of the TCP handshake to the client without having received any answer from the server. In other words, multiple TCP sessions are built asynchronously between the client and the server.

To confirm the second assumption, we checked the encryption and Session IDs of the TLS packets passing through a RESIP infrastructure. If the RESIPs broke the TLS connection, the same TLS packet sent between client and server would show different encryption and/or different Session IDs when analyzed at the client and at the server.

From a client machine, we performed TLS connections to a server passing through the four RESIP infrastructures. We analyzed the packets containing the "ClientHello" and "ServerHello" TLS records both on the client and server

[6] Previously known as Luminati.

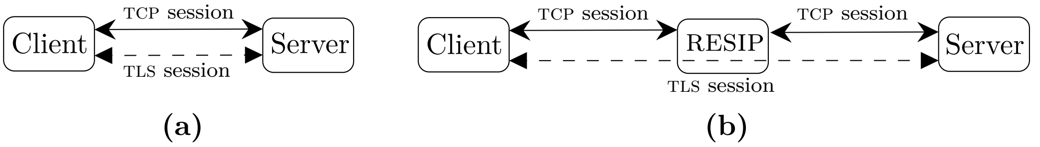


Figure 3.2: TCP and TLS sessions in a direct (a) and RESIP connection (b).

sides. We did not see any mismatch regarding encryption and Session IDs between the packets sent from one party and the corresponding ones received by the other party. This confirms that the TLS packets are not decrypted and re-encrypted by any of the machines on the path.

These results tell us that in the studied scenario, the TCP session is broken while the TLS one is not. It is possible that RESIPs are not the cause of this situation. Other network proxies could break the TCP session while only forwarding the TLS packets (e.g. TCP terminating proxies [95], [96]). However, all the RESIP connections we tested in this preliminary test, as well as the 69M+ of the experiment described in Section 4.2, result in this outcome. This enables us to make the reasonable statement that, even if there exist multiple proxies able to produce this situation, RESIPs are the most likely explanation for this result.

3.4 Detecting RESIPs with Round Trip Time Measurements

As explained in the previous sections, when a client behind a RESIP connects to a server, the client creates a TCP session with the SUPERPROXY. Then, the GATEWAY creates a different TCP session with the server. These two sessions are not synchronized. By contrast, there is only one unique TLS session built between the client and the server.

In the case of a direct connection between a client and a server, both TCP and TLS sessions take place between the same parties. Fig. 3.2 displays this difference between the two types of connections.

Let us consider what happens when a server sends a TCP packet[7] and a TLS one to a client behind a RESIP. The TCP packet has a much shorter journey than the TLS one. While the TCP packet stops at the GATEWAY, the TLS one arrives at the same machine, traverses the RESIP infrastructure, reaches the SUPERPROXY and then arrives at the client. Moreover, client, GATEWAY and SUPERPROXY could be far from each other, increasing the difference in distance traveled by the TLS packet in comparison to the TCP one.

Let us now consider a scenario in which no RESIP is involved. In this scenario, we make the reasonable assumption that there is no route change between different packets of the same connection that are sent in a short amount of time. Moreover, we assume that, if the route changes, the length of the new path is comparable to the previous one. In both these cases, there is no noticeable difference in the distances traveled by the TCP and TLS packets.

We can exploit this difference at the transport layer to identify RESIP connections reaching a server. Our intuition consists in leveraging the distances traveled by TCP and TLS packets. If the difference is high, we infer that the connection traversed a RESIP infrastructure. Otherwise, we consider it as coming directly from a client.

On the server side, we can identify this difference thanks to the Round Trip Time (RTT). The RTT measures the time between when the server sends a packet A to the other machine in the session and when packet B, which is the answer to packet A, arrives at the server. Thus, the RTT, a measure of time, exhibits a higher (resp. lower) value if a packet travels a longer (resp. shorter) distance.

We can then consider the RTT as an approximation of the measure of distance between the two parties involved in a session[8]. For direct connections, the RTT$_{TCP}$ informs about the distance between the client and the server. In the case of a RESIP connection, this value represents the distance between a RESIP GATEWAY and a server. Indeed, since the TCP sessions are not synchronized, the GATEWAY answers to the TCP packets before receiving any other packet from the client.

In both direct and RESIP scenarios, the RTT$_{TLS}$ represents the distance between a client and a server. When scrapers use RESIP services, the TLS packets take a much longer journey than the TCP ones. Hence, we expect the RTT$_{TLS}$ at the TLS layer (RTT$_{TLS}$) to be much larger than the one at the TCP layer (RTT$_{TCP}$).

We can check the δ_{RTT}, defined as the difference between the RTT$_{TLS}$ and the RTT$_{TCP}$, of each connection. If the δ_{RTT} presents a value above a chosen threshold, we detect it as proxied through a RESIP. We call this technique RTT_DETECTION.

Naturally, the path taken by the packets is not the only factor that

[7] In the rest of this work, for the sake of conciseness, we write "TLS packets" for the TLS packets generated by and exchanged between the client and the server. We write "TCP packets" for the TCP packets generated and exchanged between a GATEWAY and a server. In case of a direct connection, the GATEWAY corresponds to the client itself.

[8] For the sake of concision, in the rest of the paper we take the liberty of using the expression "measure of distance" instead of "approximation of the measure of distance" when referring to the RTT.

influences the RTT. Client processing time, packet speed, network delays, server TLS version and other factors could impact this measurement. Hence, we need to test this technique on connections between real machines on the Internet to be able to assess the validity of the method and the extent of the impact of these factors.

Moreover, other proxies in the network behave as RESIPs at the transport layer, as mentioned in Section 3.3. In this case, the designed detection method categorizes them as RESIP. It is important to understand in which percentage of connections this scenario takes place and if we can tune the method to avoid false positives.

We have designed experiments to test the validity of our method and assess the impact of the above-mentioned factors. Chapter 4 presents the setup and discusses the results of these experiments.

The designed detection technique presents some limitations. The key concept of the technique is measuring RTTs at the TCP and TLS layers. Hence, an intrinsic limitation of the technique is to work only for HTTPS connections. One easy way for scrapers behind RESIP to evade detection is to downgrade the connection to HTTP. For this reason, HTTP connections should not be allowed when using the detection technique.

The RTT_DETECTION assumes that RESIPs do not break the TLS session at the GATEWAY. By all means, RESIPs could evade detection by producing this break. This scenario is technically feasible but implies a substantial change in the RESIP model.

To break the TLS session at the GATEWAY, RESIPs need to establish a TLS connection between GATEWAY and server and another one between GATEWAY and client. This last session could be broken into two additional sessions, one between client and SUPERPROXY and one between SUPERPROXY and GATEWAY. In any case, the break implies that the GATEWAY must handle two distinct TLS connections and decrypt/re-encrypt in both directions.

RESIP providers leverage as GATEWAYs home devices, mobile phones, etc. which they do not own and, thus, they do not control. One effect of breaking the TLS session is that GATEWAYs could monitor and/or modify the exchanges between the clients and the servers. It is improbable that RESIP customers would let a third party observe and possibly modify the contents of their communications since this could damage their web scraping activity.

Furthermore, RESIP software on GATEWAYs must consume as few resources as possible to remain unnoticed by the owners of these devices. Users accept (consciously or not) to run RESIP software on their machines when they do not need them (e.g. when they are not using them and they are being charged). The additional burden of having to manage two distinct TLS sessions plus the decryption/re-encryption in both of them is likely to be a deterrent for device owners.

Lastly, to be able to impersonate any possible end server, the RESIP needs to establish certificates on the fly and push them to the GATEWAY. Moreover, the client needs to accept a RESIP-onwed root certificate to make this possible. While technically feasible, this would add a level of complexity and latency that would hurt the RESIP business and would be a significant threat to the client once this root certificate is installed.

For all these reasons, we believe RESIP would unlikely break the TLS session. If RESIP providers were to do that, we could still measure the RTT_{TLS} to the end client by serving specifically crafted HTML pages. These pages would contain objects that the client needs to retrieve from the server. We define these objects in a way that RESIPs can not cache them. In this scenario, we could obtain an approximation of the RTT_{TLS} by measuring the time between the sending of the page to the client and the arrival of the request for the additional object. This solution is similar to solutions already proposed to identify generic proxies [51]–[54]. To defeat this new scenario, RESIPs would need to run a browser-like application on the GATEWAY itself. This would increase even more the complexity of RESIP systems and for this reason, this solution is unlikely to be implemented.

Finally, the RTT_DETECTION assumes that GATEWAYs build a TCP session with the server that is not synchronized with the one between client and SUPERPROXY. We could wonder whether RESIP providers could produce delays at the TCP layer in the GATEWAY-server session to synchronize with the client-SUPERPROXY one. However, this operation is not feasible. RESIP providers do not own the devices to proxy the requests. They simply leverage them at the application level. Hence, they cannot alter the connection settings of the device and/or do kernel-level modifications to increase the RTT_{TCP}.

3.5 Residential IP Proxies Identification based on Retransmission Protocols

In Section 3.3 we performed tests to confirm two working assumptions about the internal functioning of RESIP providers. In the first test, a client initiates a connection, passing through a RESIP infrastructure, to a server not sending any ACK packets. Generally, when a client sends a SYN packet and this is not acknowledged, its kernel keeps retransmitting the SYN packet from the same port using the exponential backoff behavior of the retransmission timeout. This continues until a higher timeout threshold is reached [97].

In the case of a RESIP connection to such server, the GATEWAY sends a

SYN packet and the server does not acknowledge it. We observed that, after a delay, new GATEWAYs, machines different from the original one, start to send SYN packets to the server. This results in multiple GATEWAYs trying to connect to the server. Our intuition is that the RESIPs believe the chosen GATEWAY has a connectivity problem. They then try to perform the request with other machines in their network.

The multiple GATEWAY connections can be sequential or partially parallel. In the first scenario, a new machine starts to query the server only after the previous machine has ended sending packets. In the second one, there is an overlapping window of time in which two machines, used by the same RESIP provider in the same communication, send SYN packets to the server. We believe that this second scenario takes place when providers try to improve their response time to the client. In the first case, the provider waits for a GATEWAY to terminate all attempted connections before choosing another machine. In the second one, there is a fixed delay after which, even if the chosen GATEWAY has not terminated yet its attempts, the RESIP provider chooses a new GATEWAY to perform the same operation. Section 5.4 shows in which percentage these approaches take place in the connections of the considered providers.

Moreover, independently of the above-mentioned behavior, each GATEWAY machine performs some kind of retransmission. Every time a GATEWAY sends one packet, it can act using one of the following Machine Retransmission Protocols (MRPs):

First MRP: The kernel of the GATEWAY resends the original SYN packet using the same port with exponential backoff, as in the normal case.

Second MRP: The GATEWAY opens a new connection to the server from a different port and the server receives a new SYN packet. Moreover, in some cases, the kernel retransmits once the original SYN packet.

Section 5.4 explains which providers, among the tested ones, expose each MRP. Our intuition is that, in the second MRP, the application layer (not the transport one) handles the sending of packets and the timeouts. When the timeout expires, the application closes the previously used socket and opens a new one. In some cases, this can produce a normal retransmission since the application timeout can be longer or comparable to the transport layer one.

The behavior shown by RESIPs in the second MRP is specific and peculiar. We can exploit it to perform server-side RESIP detection of the providers that adopt it. The idea is to detect if a machine sends a new SYN packet from a different port when the original one is not acknowledged in time. We call this approach MRP_DETECTION.

To perform this detection, the server simply needs to delay the sending of the SYN-ACK packets. In this way, it can check if it receives other requests from the same IP that are not retransmissions from the same port. To achieve this result, the server can take advantage of a shared structure in memory where all the threads that handle each connection save the timestamp, IP and port of each received SYN packet. If the same IP sends SYN packets from different ports in a short amount of time, the connection likely goes through a RESIP provider adopting the second MRP. The server can then decide to never send a SYN-ACK packet in response to those SYN packets. In this way, it provides mitigation on top of detection. Indeed, the RESIPs waste resources, having different devices in their network trying to connect, uselessly.

The above-mentioned technique presents some limitations. A server implementing it needs to delay the answer to each SYN packet for enough time to potentially receive a new SYN packet from the same machine. This duration depends on how the providers implement the MRP. If this duration was long, genuine clients would experience high delays. This would degrade the real users' experience. Section 5.3 displays the interval between new SYN packets for the analyzed providers that use this strategy and Section 5.4 discusses if the corresponding delay is acceptable at the client side.

Furthermore, RESIP providers could avoid this detection by changing their MRP. They could randomly select different values for the interval after which a machine sends a new SYN packet. Moreover, they could switch to the first MRP. Depending on their setup, this could be a difficult task to achieve and could imply a loss in efficiency and costs on their side.

The MRP_DETECTION technique results in false positives if some corner cases take place. If two machines behind the same NATting device contact the same server in a short amount of time, the technique categorizes their connections as RESIP. Moreover, this outcome occurs if a genuine user requests the same URL two or more times from the same device in a short amount of time. While these events are possible, we find them unlikely to happen on a large scale.

Finally, the proposed approach is only able to identify specific RESIP providers that adopt the discussed MRP. While the technique is less general than the one described in the previous section, it enables us to link scraping campaigns to specific RESIP providers and thus perform attribution.

3.6 Summary

In this chapter, we have learned that RESIP infrastructures are a fixed environment that scrapers cannot modify. Hence, we can exploit specific features of such environments to understand, server-side, if an incoming request is proxied through them. Based on this, we proposed two new detection techniques and we explain the underlying intuition for them.

The first approach (RTT_DETECTION) detects RESIP connections thanks to Round Trip Time differences at the TCP and TLS layers. The second one (MRP_DETECTION) identifies RESIP connections when they adopt a specific Machine Retransmission Protocol. It consists of a retransmission timeout handled at the application layer and not at the transport one.

The possibility exists that those ideas, simple and straightforward in theory, could not work in practice because of technical limitations, as discussed in the sections dedicated to each technique. This is why we have run long and thorough experiments to assess their feasibility and validity. In the next chapters, we outline the setup of such campaigns and the results we obtained.

Chapter 4

Round Trip Time Measurements to Identify Scrapers behind Residential IP Proxies

4.1 Introduction

In Section 3.4, we outlined the underlying intuitions for the RTT_DETECTION technique. RESIP providers break the TCP session but not the TLS one. Thanks to this, we can identify RESIP connections by comparing the Round Trip Times (RTTs) at the two layers.

The same section points out that, while the idea is sound in theory, it could be impacted by different factors in practice. We designed a series of experiments to understand the validity of the approach and understand the extent of the different impacts. Hereafter, we present their setups and we discuss their results.

Section 4.2 introduces the setup of our first experiment, Ground Truth Experiment. This experiment aims at assessing if the difference between the RTT$_{TLS}$ and the RTT$_{TCP}$ is systematically, constantly, and significantly higher for RESIP connections than direct ones when the same client is used in both scenarios. In this experiment, we know, by design, which connections are proxied through a RESIP and which ones are not. Section 4.3 presents the analyses performed on the dataset of Ground Truth Experiment and assesses the validity of the RTT_DETECTION technique.

The choice of specific clients in the generation and transmission of packets (e.g. using a web browser, hotspot connections, VPN usage) could impact

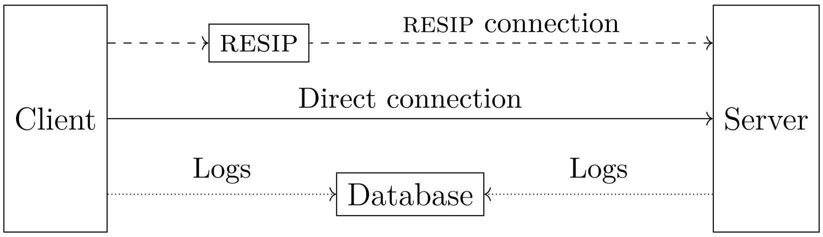


Figure 4.1: Infrastructure used to validate the RTT_DETECTION technique.

the technique. Hence, non RESIP connections could be wrongly categorized as RESIP ones. In Section 4.4, we show which factors produce this situation characterizing different client-side environments. Furthermore, we estimate the probability of facing such situations by applying the RTT_DETECTION method on real-world connections. Finally, we investigate how many non RESIP connections are categorized as such because of Mobile TCP Terminating Proxies used by ISPs. We bring evidence that by highering the threshold, we could avoid a large part of these false positive cases.

The underlying idea of the RTT_DETECTION technique, the setup of the Ground Truth Experiment and analyses of its dataset to evaluate the validity of the technique were published in [C2]. Additional analyses of the dataset (Section 4.3.1, Section 4.3.4) were presented in [C3]. Results in Section 4.4 were published in [C4].

4.2 Ground Truth Experiment Setup

In this section, we present the setup of the Ground Truth Experiment. Its main goal is to understand if the δ_{RTT} is systematically, constantly, and significantly higher for RESIP connections than direct ones when the same client is used in both scenarios. Moreover, this experiment enables us to assess the impact on the technique of network delays, proximity between RESIP machine and our machines, TLS version run at the server and packet speeds.

As previously explained in Section 3.2, RESIP infrastructures are quite specific. To achieve the above-mentioned goal, we have created an infrastructure that reproduces the real-world conditions scrapers experience when using such services. Section 4.3.1 provides evidence of such a statement.

As shown in Fig. 4.1, our infrastructure includes a client sending requests to a target server, a target server, a RESIP provider, and a database.

The client, which represents the scraper, sends requests either directly to the server (direct connection) or through a RESIP provider (RESIP connection)

Table 4.1: Monthly subscriptions of the RESIP providers used in the experiment.

Provider	**Plan**	**Traffic**	**Price per GB**	**Advertised Pool Size**
Bright Data	Starter	40GB	12.5$	72M+
Oxylabs	Business	50GB	12$	100M+
Proxyrack	Premium Residential Proxies	50GB	4$	5M+
Smartproxy	Regular	50GB	8$	40M+

from which we purchased the services. On the other side of the connection, the server analyzes each received query. Furthermore, on the server machine, there is a sniffer that looks at every received packet and performs network measurements. The client and the server machines locally produce logs and send them to a database where they are aggregated and processed.

22 machines and 4 RESIP providers constitute the core of the infrastructure. In the next sections, we discuss their details. Section 4.2.1 discusses the examined RESIP services. Section 4.2.2 explains the configuration, location, and roles of our client and server machines. Section 4.2.3 outlines the performed network measurements for the detection technique. Section 4.2.4 describes the timeline and data storage of the experiment.

4.2.1 Residential IP Proxy Providers

In this experiment, we test 4 RESIP providers widely used, allegedly, by scrapers: Bright Data [91], Oxylabs [92], Proxyrack [93], and Smartproxy [94]. We chose them thanks to the information provided by analysts and companies working against scraping bots as well as online blogs devoted to web scraping activities. These are the same providers used in Section 3.3 to test the working assumptions.

The four services offer different packages and options. We subscribed to each of them to have 40GB (Bright Data) and 50GB (the other providers) of (incoming+outgoing) traffic per month proxied through residential IP addresses. Table 4.1 provides detailed information about our subscriptions. We set the IPs to be changed at each new request and we did not impose any localization constraint for them.

Table 4.2: Technical description of the machines in the experiment.

Provider	OS	Memory	Processing	Storage
Amazon Lightsail	Ubuntu 20.04	2GB	1vCPU	60 GB SSD
Azure	Ubuntu 20.04	3.5GB	1vCPU	300GB+ SSD

We legitimately made and paid our subscriptions for the four services online. Three out of four providers gave us access to the pool of residential IPs upon payment.

By contrast, Bright Data did not enable us to use the residential IPs directly after the payment. They asked us to participate in a recorded interview in which we had to explain the motivations of our work and how we wanted to use their infrastructure. We communicated to them that we wanted to test our client and server machines with their infrastructure. We told them that we would simply perform requests from our client to our server machines, as we did.

However, 13 days after the beginning of the experiment, they paused our subscription telling us that our scenario (targeting our own machines) could "expose their users IPs, which can become a privacy issue"[98]. They told us that we would need to disclose additional information about what we were doing. Since we did not agree to do so, they completely stopped the subscription and refunded it.

4.2.2 Clients and Servers

The core of the infrastructure consists of 22 machines. Each of these machines plays both the roles of client and server described in Fig. 4.1. In this way, we maximize the number of different client-server paths available for the experiment. To avoid any possible geographical or vendor bias, client and server machines are spread all over the world and belong to two different suppliers: Amazon Lightsail [99] (16 machines) and Microsoft Azure [100] (6 machines).

Each of the following locations hosts two machines: India, Australia, Japan, Germany, Ireland, Canada, USA (Virginia and Oregon), South Africa, United Arab Emirates and Brazil. The last three locations correspond to the machines acquired from Azure. Table 4.2 shows the technical details of the machines used in the experiment.

We implemented both client and server algorithms in Python3 [101]. The server code leverages the ThreadingHTTPServer and BaseHTTPRequestHandle objects of the library http.server [102]. We modified the source code of this library to insert a timeout for connections not completing the TCP handshake. The client performs both direct and RESIP connections using the library urllib [103].

The client algorithm consists of an infinite loop. According to the speed set in the configuration file, each client sends queries to each server in the experiment. Each client queries five times consecutively each machine, with one direct connection and four RESIP ones, one per provider. The query consists of an HTTPS GET.

To provide the server with all the information about each connection it receives, we encode the client IP address and the RESIP provider identifier into the requested URL. We assign a unique two-digit code (machine code) to each machine, from 01 to 22, and a one-digit code (RESIP code) to each RESIP provider, from 1 to 4. Direct connections have a one-digit code (direct code) set to 0. Every time the client performs a request, it builds the URL as the concatenation of its machine code, the RESIP/direct code, and the target server machine code.

The server algorithm keeps listening for new incoming HTTPS connections on port 443. The server can run both TLS1.2 and TLS1.3. At launch time, we can define an option to specify which version to run. Every time a new HTTPS GET request arrives, the server studies the URL. It verifies that the request format is compatible with the concatenation of the codes. In this way, it assures that the request originates from a machine participating in the experiment (as opposed to, eg. a scanner or a crawler). It then retrieves the client, RESIP/direct and server codes. The server answers with an error page to requests that do not pass the check. Otherwise, it delivers a simple page.

Both client and server programs locally log the connections. These logs are aggregated in a database. In Section 4.2.4 we present an overview of the collected data and we explain which field we logged on each machine.

4.2.3 Network Measurements

A sniffer program runs on each machine of our infrastructure to collect network measurements. The sniffer parses each incoming and outgoing packet to port 443. We implemented the sniffer in Python3, thanks to the library PyShark [104]. The sniffer saves the information about packets in structures representing the corresponding streams.

During our experiment, we noticed that Pyshark was exhibiting singular behaviors when running for a long time. This was caused by the fact that Pyshark live captures do not have an efficient garbage collection mechanism. This can lead to memory problems. Unfinished connections produced by

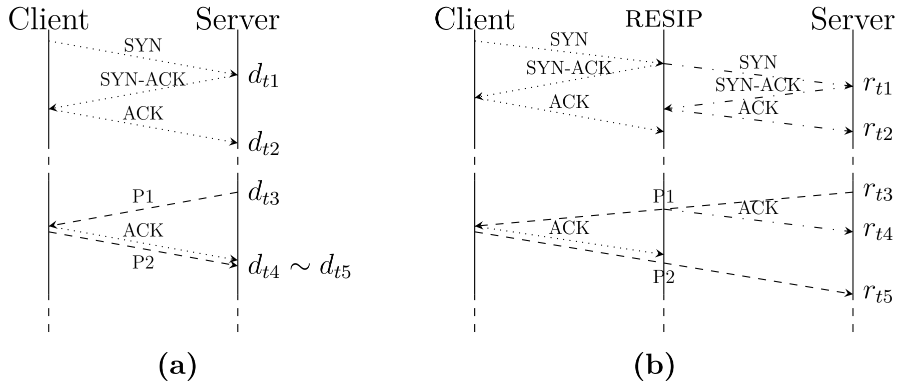


Figure 4.2: TCP and TLS packet exchanges used in the RTT_DETECTION technique for a) direct and b) RESIP connections.

intense scanning of our machines exacerbate this problem. To avoid these singular behaviors and preserve our measurements, we run the sniffer for just a short amount of time and restart it systematically every hour. Section 4.2.4 discusses the effects of this choice on the data collection.

For each incoming stream, we use the RTT to understand how far from each other the parties taking part in the communication are. As discussed in Section 3.4, RTT$_{TCP}$ gives us the distance between client and server, for direct connections. In the case of a RESIP connection, this value represents the distance between a RESIP GATEWAY and a server instead of between the original client and the server. In both scenarios, the RTT$_{TLS}$ represents the distance between the original client and a server. If the difference between the RTT$_{TLS}$ and the RTT$_{TCP}$ (δ_{RTT}) is higher than a chosen threshold, we declare the connection as passing through a RESIP.

Hence, we need to calculate two RTTs, RTT$_{TCP}$ and RTT$_{TLS}$. Fig. 4.2 helps to explain these measurements. Fig. 4.2a shows the packets in the TCP (dotted lines) and the TLS (dashed lines) exchanges that we use for detection in case of direct connection. Fig. 4.2b presents the same exchanges in the case of a RESIP connection. Dotted lines represent TCP exchanges between the client and the RESIP, dash-dotted lines stand for TCP exchanges between RESIP and server and dashed lines show the TLS exchanges. d_{tx} and r_{tx} represent the timestamp measurements of the sending/arrival of a packet at the server for direct (d_{tx}) and RESIP (r_{tx}) connections. Hereafter, we define each of the two measurements.

RTT$_{TCP}$

The RTT$_{TCP}$ is the RTT between the SYN-ACK packet sent by the server and the corresponding received ACK. In the case of direct connection, the client creates the TCP connection directly with the server. Thus, the RTT$_{TCP}$ (d_{t2}-d_{t1} in Fig. 4.2a) is a measure of the distance between these two parties.

By contrast, in a RESIP connection, the RESIP builds two distinct TCP connections. One connection takes place between the client and the SUPER-PROXY, and one between the GATEWAY and the server. In this scenario, the RTT$_{TCP}$ (r_{t2}-r_{t1} in Fig. 4.2b) represents the distance between GATEWAY and server.

Throughout the lifetime of a given TCP connection, the network conditions can vary. There is uncertainty that the RTT calculated at the establishment of the connection is representative of the "real" RTT$_{TCP}$ for that connection. To understand how this variability can influence our analysis, we collected the RTT for all TCP packets sent by the server, in addition to the very first RTT$_{TCP}$ value described above. We calculate statistics and discuss them in Section 4.3.3.

RTT$_{TLS}$

The RTT$_{TLS}$ corresponds to the RTT of the TLS layer. The TLS protocol is end-to-end between client and server both in case of direct and RESIP connections. Thus, this metric should give us the measure of the distance between client and server in all scenarios.

To obtain the RTT$_{TLS}$, we consider two packets, P1 and P2. P1 contains a server TLS record after which the server does not send any other TLS record before receiving a specific TLS answer, as per the protocol. P2 is the packet containing the client TLS record that allows the continuation of the protocol. We can choose any couple of TLS packets that satisfies these conditions.

As explained in Section 3.2, we can consider the RESIP architecture fixed for our scenario. Thus, the only variables that can influence our choice of P1 and P2 are the server and client implementations.

Our detection method is server-side. We assume anyone recreating this experiment has full access and knowledge of the server implementation. In the TLS connection, the server dictates the rules of the exchanges e.g. accepting the cipher proposed by the client or deciding if it requires client authentication. In such conditions, we expect to be able to anticipate all the possible exchanges between client and server to find a couple of packets and cover possible corner cases. For these reasons, having full control over the client in our experiment, contrary to the real-world case, does not constitute a bias.

In our setup, we have a generic HTTPS server, which does not require any client authentication and accepts common encryption ciphers. We expect this to be a common scenario for scraped websites that need to be accessed by the largest possible number of clients. In these conditions, we can identify P1 and P2 among the first TLS packets. This is an added value because it enables us to perform detection before any application content is delivered to the client.

Hereafter, we focus on the TLS records we use to perform our measurement. Depending on the TLS version, we use different records to identify P1 and P2. We refer to [105] for an accurate and detailed description of the TLS protocols and the records that are not involved in our measurement.

In TLS1.2, we identify P1 and P2 in the TLS handshake where all messages are in clear text. The RFC 5246 [106] states that, after sending the "ServerHelloDone" TLS message as part of the TLS handshake, the server waits for a client response. We recognize P1 as the packet containing the TLS record encapsulating this message.

After the "ServerHelloDone" message, the client needs to continue the communication. According to the RFC, it must send as first TLS message the "ClientKeyExchange" one. We identify P2 as the packet whose TLS record contains this TLS message.

In TLS1.3, content encryption starts in the TLS handshake. If the server agrees on the cipher chosen by the client, it sends the "ServerHello" TLS message. Since the server has already obtained the client-side information for encryption, the data in the message is encrypted.

As explained in Appendix D of RFC 8446 [107], TLS1.3 implementations include a dummy "change_cipher_spec" TLS record to guarantee backward compatibility for middleboxes. The client sends this record before its encrypted handshake flight if, by setup, the client does not send early data and if it does not send a second "ClientHello" message. In our implementation, we do not offer early data. Moreover, the server accepts all common ciphers, among which the one used by the client. Thus, the server does not send the "HelloRetryRequest" TLS message that forces the client to send a second "ClientHello" message[9]. Hence, the "change_cipher_spec" TLS record is the first client TLS record sent upon reception of the "ServerHello" TLS message. This record is in clear text and, thus we identify P2 as the packet containing it. From this identification of P2, we look at previous packets sent by the server to find P1. More precisely, we determine P1 as the packet whose TLS

[9] In cases where the server could send the "HelloRetryRequest" TLS message, we can simply perform the measurement considering that also the packet containing this TLS message could identify P1. In that case, the next packet sent by the client (the one containing the second "ClientHello" message or the alert that aborts the handshake) would identify P2. These messages are all in clear text.

record contains the last encrypted server data sent after the transmission of the "ServerHello" message and before the arrival of P2[10].

We define the RTT_{TLS} as the difference between the sending of P1 and the arrival of P2 at the server. As shown in Fig, 4.2, this corresponds to the difference d_{t5}-d_{t3} in the case of direct connection and r_{t5}-r_{t3} in the case of RESIP one[11].

4.2.4 Timeline and Data Storage

We run the experiment from 15:00 UTC +0 on 12/01/2022 till 01/05/2022 at 15:00 UTC +0. Thus, the total number of days is 110.

Every day at 00:00 UTC +0, we restarted each server and we switched its version from TLS1.2 to TLS1.3 and vice versa[12]. In this way, we obtained the same amount of data for both protocols.

Initially, only 16 machines from Amazon were part of the experiment. Considering all machines, we sent/received 10.88 requests/second and each RESIP provider was processing 2.18 requests/second. We used these rates to remain below the limits imposed on us by our RESIP subscriptions. On 24/01/2022 at 19:00 UTC +0, we added 6 machines from Azure to our pool. At first, we kept the same rates per client/server. Hence, the rate was 14.96 requests/second and the ratio per RESIP provider was 2.99 requests/second. On 25/01/2022 at 16:00 UTC +0, Bright Data stopped our access to their network and ended our subscription. Section 4.2.1 provides the motivation for this choice. Since it was not possible to restore this service, on 02/02/2022, we eliminated it from our experiment. We adjusted the rates accordingly and since then, 9.90 requests/second were sent/received for the rest of the experiment. Each RESIP provider processed 2.48 requests/second.

On 07/03/2022 at 00:00 UTC +0, we started collecting more network information to study the variability of our measurements. For each connection, we measured the RTT of all the TCP exchanges. On 14/04/2022 at 00:00 UTC

[10] We could instrument the server to identify P1 from its decrypted content. However, we want to be as less invasive as possible, providing a network monitoring approach, and not obliging to modify the server code.

[11] The reader may wonder why we are not using the arrival of the ACK packet of P1 as the second point of measurement (d_{t4} and r_{t4}, in Fig. 4.2). In the case of a RESIP connection, there are two distinct TCP connections (client-SUPERPROXY and GATEWAY-server, as explained in Section 3.2). The kernel is in charge of creating the ACK packets and these are sent at the arrival of P1 at the GATEWAY, without synchronization with the client-SUPERPROXY TCP connection. Hence, the difference r_{t5}-r_{t4} represents the distance between GATEWAY and server and not the one between client and server.

[12] Later on in the section, we discuss the effects of this choice on the data collection.

Table 4.3: Attributes of a connection in the RTT_DS dataset

Attribute	Attribute explanation
CLIENT EPOCH	Epoch (UTC+0) in which the client sends the request [s]
CLIENT CODE	Two-digits code identifying the client starting the connection
RESIP CODE	One-digit code identifying the RESIP provider or direct connections
SERVER CODE	Two-digits code identifying the server receiving the request
SUPERPROXY IP	IP address contacted by client
GATEWAY IP	IP address the server sees as the source address of the request
TLS	TLS version run by the server at the time of the request
RTT$_{TCP}$	Measured RTT at the TCP layer (first exchange) [ms]
RTT$_{TLS}$	Measured RTT at the TLS layer (between P1 and P2) [ms]
MIN RTT$_{TCP}$	Minimum measured RTT at the TCP layer [ms]
MIN POSITION	Position in the stream of the minimum RTT at the TCP layer
MAX RTT$_{TCP}$	Maximum measured RTT at the TCP layer [ms]
MAX POSITION	Position in the stream of the maximum RTT at the TCP layer
TTL	Time-To-Live of the first client packet received by the server

+0, we began logging additional network information, the Time-To-Live (TTL) corresponding to the first client TCP packet received at the server.

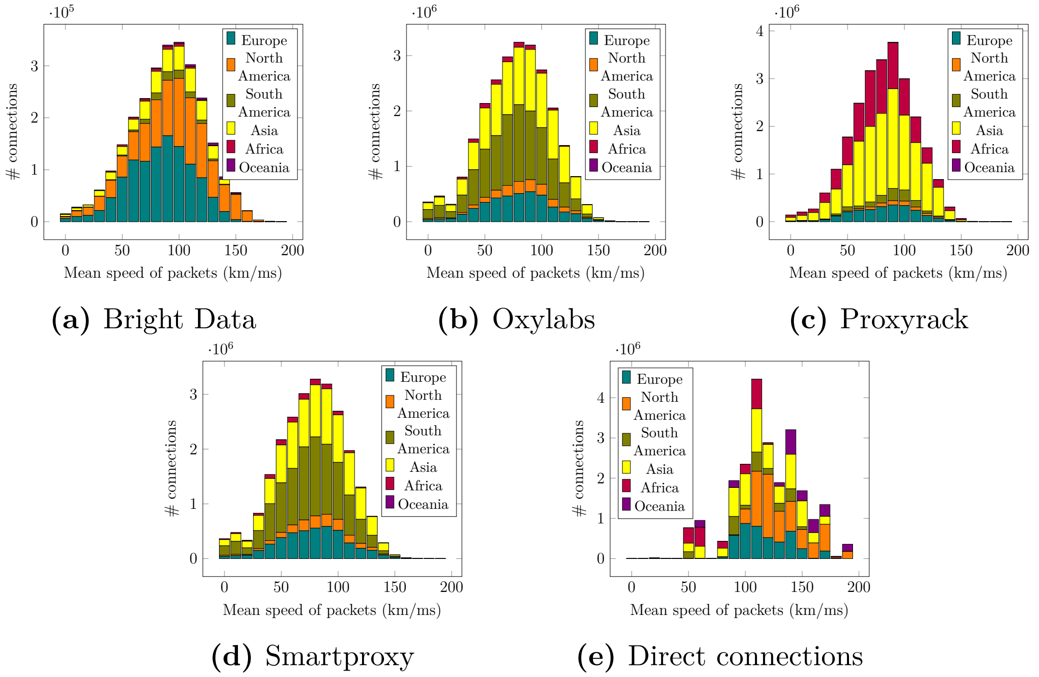
Occasionally, some machines were restarted by the cloud providers and this resulted in small losses of data. We also had some brief synchronization issues, caused by using one port for both TLS1.2 and TLS1.3 server programs and switching among them at midnight. Fortunately, this has been an extremely rare event. It has happened, on average, only 1.6 times per machine over the 110 days, with an average loss of only 0.17% of the traffic per machine.

We have created a database with POSTGRESQL [108] to gather the data from the experiment. In the database, we keep a unique record for each connection. This record includes information collected at the client and the server as well as the network measurements.

We define the collected dataset as RTT_DS. Table 4.3 presents, with the corresponding explanation, the attributes we consider for each connection in our dataset. Moreover, we enrich the information about these connections thanks to external databases. We study the geolocation of the IPs we collected thanks to the MaxMind GeoLite2 databases [70]. These databases contain information about the city, country and Autonomous System Number (ASN) of IPs. We unify all the information for our IPs in the three databases into one called GEOLOCALIZATION_DS. The accuracy of the MaxMind database has been questioned in the past [72], [73]. However, it is recognized that its data is reliable at the country level [74]–[76]. In our analyses, we are not interested in the precise location of each IP but we consider the geolocalization information at a high granularity. Thus, even if the GEOLOCALIZATION_DS has limitations in accuracy, they do not influence our results.

In Section 7.2.6, we use the TTL of each client connection to characterize the OSes of GATEWAYs, which should be residential devices, among different providers. To associate each TTL to the corresponding OS, we take advantage of FINGERPRINTS_DS. This dataset links TCP/IP fingerprinting information with the corresponding OS. Lastovicka et al. built it in [109] thanks to measurements performed in an academic wireless network and, thanks to this, it characterizes many classes of residential devices. The dataset is available at [110].

In total, our clients have generated close to 98M connections but, for the reason explained before, some observed connections were incomplete. A client querying a server when this last machine was down produced a logged request only on the client side (around 4M). Similarly, machines not part of the experiment (e.g. scanners and crawlers) sent requests to our servers and, for those, we have no matching record in the client logs (around 200K). Moreover, the sniffer program restarts every hour, and incoming connections arriving at the moment of the switch could have incomplete RTT measurements. We only create a record in our database for connections that exist in both the client and server logs and for which we have at least the measurement of the RTT$_{TCP}$ of the first exchange and the measurement of the RTT$_{TLS}$. As a result, we use 95% of the total amount of connections started by the clients which sum up to 92,712,461 connections (9.76 entries/second for 110 days). In the next section, thanks to these connections, we show that our detection technique is able to detect when scrapers exploit RESIP services. Moreover, in Chapter 7, we provide IP-based analyses of the same connections. Thanks to this examination, we introduce some novel findings about the inner workings of RESIP providers.



(a) Bright Data **(b)** Oxylabs **(c)** Proxyrack

(d) Smartproxy **(e)** Direct connections

Figure 4.3: Distribution of the mean speed of packets for each RESIP provider and direct connections.

4.3 Analyses of the Ground Truth Experiment Dataset

4.3.1 Packet Average Speed Analysis

We designed our experiment to be as close as possible to a real-world scenario in which a scraper sends requests behind a RESIP infrastructure. To achieve this goal, we use machines located around the world and acquired from different providers. Moreover, we use RESIP services commonly exploited by scrapers and we perform connections using the public Internet.

Our detection technique is based on the RTT mirroring the distance between the parties involved in a connection[13]. However, the RTT is a measure of time. To transform it in a distance, we need to consider the speed at which packets move. If this speed had a common mean value, there would be a proportional factor between each RTT value and the corresponding traveled distance of a packet. This factor would be common to all the connections and this would favor the success of our detection technique. However, the real world does

[13] We assume the processing time at the client to be negligible with respect to the transmission time.

not present perfect conditions. As acknowledged by Weinberg et al. [87], an idealized common value for the average speed does not exist in practice for connections across different areas of the world.

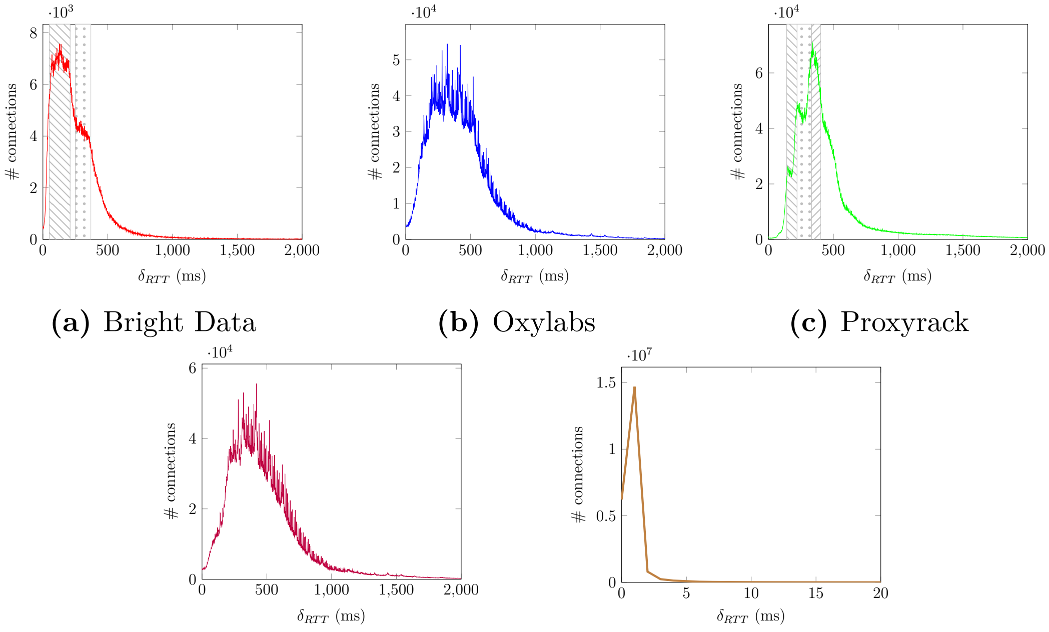
This section highlights the great diversity of the hypothetical average speed of packets. This convinces us that the results presented afterward are not only valid for certain kinds of "well-behaving" connections. Instead, our results have been obtained on a wide range of operational network conditions and are thus representative.

We propose a study of the average speed of packets between our GATEWAYs and servers. The definition of average speed is the ratio between space and time. The space, in our context, is the distance between each GATEWAY and a server. We acquire the location of each GATEWAY thanks to GEOLOCALIZATION_DS. We calculate the distance between each GATEWAY and a server thanks to the Haversine distance [111], which approximates the distance on Earth between two points given their coordinates. We consider the time from a GATEWAY to a server by dividing by half the RTT_{TCP} recorded in the corresponding connection.

We analyze individually the connections of each RESIP provider. For each provider, we consider all the RTT_{TCP} measurements acquired between each server and each GATEWAY that sent requests to it. Moreover, we perform the same analysis for the direct connections between our clients and servers. We exclude the combinations of client and server in which the machines are in the same location. We cannot calculate the distance between them since they share the same coordinates. We use, once again, the Haversine formula to obtain the distance between each client and each server. We collect all the direct connections among each couple of machines and we calculate the time as half of the RTT_{TCP} of the connection.

Fig. 4.3 shows the histograms of the obtained results. On the x-axis, we see the average speed of packets in km/ms. On the y-axis, we find the number of connections showing that speed in the dataset. The average speeds are shown in different colors depending on the continent in which the sender is located. In this figure, we do not consider the cases where the value of the speed is higher than 200km/ms, for better visualization, or where the continent is not available in the database. These two cases together correspond to 0.08% of all the considered connections and thus we consider their contribution negligible.

We can see how the distribution of the speed for each RESIP provider (Fig. 4.3a-4.3d) ranges from 0 to 150km/ms. Moreover, we see that the distribution for the connections from GATEWAYs of each continent follows a shape comparable to the global one. These curves highlight the great variability of the average speed of packets in our monitored connections. Our dataset is clearly representative of the real conditions of the Internet in which



(a) Bright Data **(b)** Oxylabs **(c)** Proxyrack

(d) Smartproxy **(e)** Direct connections

Figure 4.4: δ_{RTT} distribution for each RESIP provider (interval 0-2000ms) and direct connections (interval 0-20ms).

multiple factors affect the time it takes for a packet to reach its destination.

Fig. 4.3e illustrates the results for the direct connections. We can see that the mean speed value ranges between different values as in the previous cases. However, the values in the range are higher in this second study. Connections coming from machines in North America only present values above 80km/ms. Connections from European locations have mostly values between 75 and 150 km/ms. The mean speed values have more variability for the connections coming from other continents. This analysis tells us that the connections between our client and servers have better connectivity levels than the ones between GATEWAYs and servers. We expected these results since the machines are located in well-connected data centers, as opposed to GATEWAYs machines, which are in unknown conditions. However, even in this case, the mean speeds do not end up with a single value. This enables us to say that our setup has no apriori bias that could compromise our experiment.

4.3.2 δ_{RTT} Values Distribution

In Fig. 4.4, we show the δ_{RTT} values for each proxy and for direct connections. On the x-axis, we see the δ_{RTT} value in milliseconds of a connection. On the

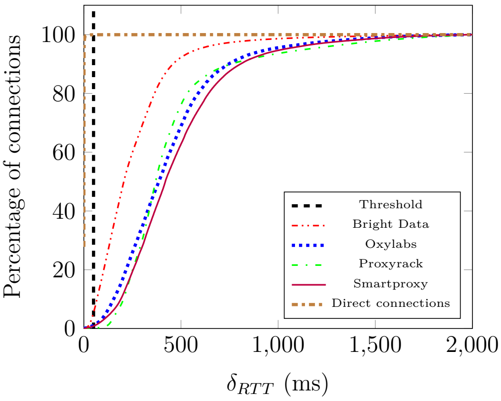


Figure 4.5: Cumulative Distribution Function of the δ_{RTT} of the RESIP and direct connections for the intervals 0-2000ms.

Table 4.4: Percentages of connections for different δ_{RTT} values (DC = Direct Connections, BD = Bright Data, OL = Oxylabs, PR = Proxyrack, SM = Smartproxy).

δ_{RTT}	DC	BD	OL	PR	SM
25	99.91	1.03	0.47	0.06	0.35
50	99.97	5.45	1.11	0.13	0.84
75	99.98	12.17	2.18	0.25	1.71

y-axis, we find the number of connections that have that specific value of δ_{RTT}. Areas in the plots with different backgrounds are highlighted to better understand the analysis presented in Section 4.3.6.

To better visualize the δ_{RTT} values of the majority of the connections, we consider different x-axis ranges. 97% of direct connections have an δ_{RTT} value lower than or equal to 20ms. In Fig. 4.4e, we use the range [0,20]ms for the x-axis of these connections. For RESIP connections (Figs. 4.4a-4.4d), we consider instead the RTT differences in the range [0,2000] ms, which amounts to the same percentage of connections.

We can see how for direct connections (Fig. 4.4e), the difference is always close to zero. In the RESIP plots (Figs. 4.4a-4.4d), instead, we can see how the difference varies for RESIP connections. It is very important to note that the maximum value of the δ_{RTT} (x-axis) in Fig. 4.4e is 100 times smaller than the ones in the other graphs. The maximum value on the y-axis is at least 3 orders of magnitude larger for direct connections than for RESIP ones. Yet, the total amount of connections is similar[14]. These results clearly show that direct and RESIP connections have dramatically different distributions of δ_{RTT}.

We need to choose a threshold above which we can safely categorize a connection as a RESIP one. To find this value, we need to solve an optimization problem where we want to minimize both the False Positive (FPR)[15] and

[14] Except for Bright Data for which we have less traffic due to the early end of the service, as explained in Section 4.2.1.

[15] Direct connections flagged as RESIP ones over the total amount of direct connections.

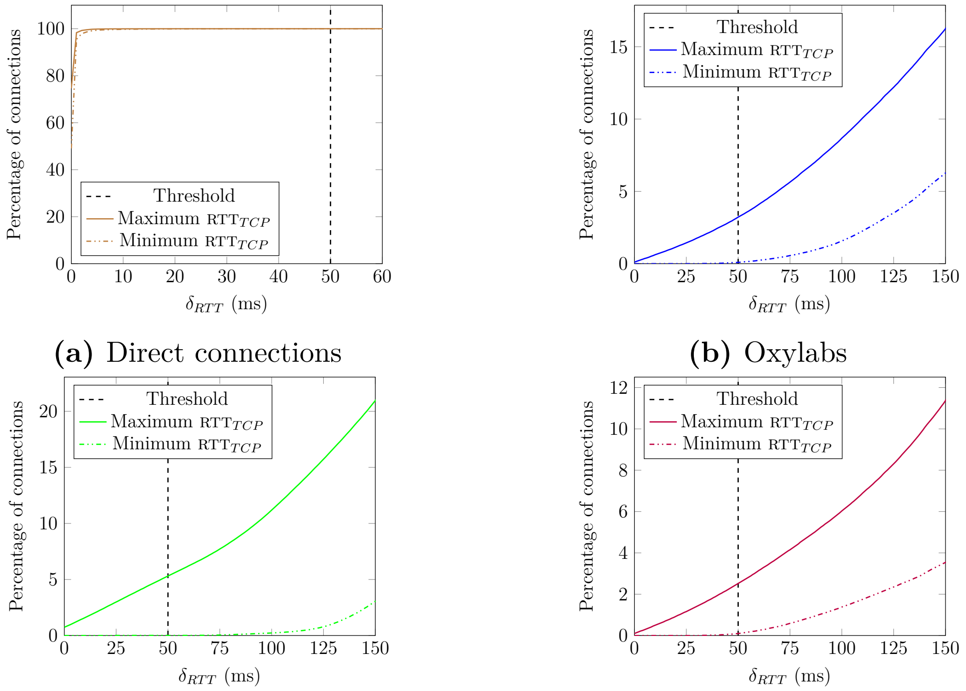
False Negative (FNR)[16] rates. Our technique aims to be used by e-commerce websites trying to stop the scraping of their content. The goal of these websites is to maintain the FPR as low as possible while doing so. To mirror this prevalence, we privilege the minimization of the FPR above the one of the FNR. However, we set as a constraint that the FNR value should not be higher than 2%, to maintain the detection efficiency in stopping scrapers.

After experimenting with different values, we can conclude that 50ms is the best value for our threshold. Indeed, choosing this threshold we obtain a FPR of 0.04% and a FNR of 1.93%. The corresponding accuracy, calculated as the number of correctly classified connections on the total amount, is 99.01%.

Fig. 4.5 helps us visualize the difference between direct and RESIP connections with respect to the 50ms threshold. It shows the Cumulative Distribution Function (CDF) of the δ_{RTT} of the RESIP and direct connections. The x-axis range shows the δ_{RTT} (in the same range as in Fig. 4.4), the y-axis the percentage of connections. A black dashed line shows the chosen value for the threshold. We can clearly see that the CDF of direct connection reaches almost 100% before the threshold. On the other hand, the CDFs of the RESIP providers grow much slower and have most of their connections way above the threshold.

Table 4.4 helps us better understand what happens to the different distributions near the threshold. At 25ms, the percentage of direct connections having this δ_{RTT} value or below is 99.91%. For each RESIP provider, the corresponding percentages are close to or below 1%. We see that the percentage for direct connection increases to 99.97 at 50ms. For RESIP connections and the same value of δ_{RTT}, the percentage for Bright Data significantly increases (factor of 5.29). For the other providers, the percentage is slightly above the double previous value. We can deduce that Bright Data is more efficient and stealthy than other providers. We believe that could be associated with their SUPERPROXY management (Section 7.2.4). Considering 75ms as δ_{RTT} value, the percentage for direct connections increases by just 0.01%, while RESIP percentages, especially Bright Data one, increase consistently.

This table shows us that 50ms is a good compromise value for the threshold. Almost all direct connections have a δ_{RTT} value below it, and after this value the increase in the true positive ratio is small. At the same time, the percentages of false negatives are still restrained. A higher threshold (75ms) would bring a much more consistent number of undetected RESIP connections.



(a) Direct connections

(b) Oxylabs

(c) Proxyrack

(d) Smartproxy

Figure 4.6: Cumulative Distribution Function of the δ_{RTT} (interval 0-150ms for RESIP connections and 0-60ms for direct ones) of the connections when the minimum and maximum RTT$_{TCP}$ are used to compute the δ_{RTT}.

4.3.3 Network Delays Impact

Our approach determines if a connection passes through a RESIP provider by measuring the δ_{RTT}. This measurement is conducted on packets sent and received on the Internet. Thus, network delays could, possibly, negatively impact our approach.

In our dataset, we see connections with negative values for the δ_{RTT}. The percentage of the total amount of connections per provider is 2.9%, 0.9%, 1.8%, 0.2%, 1.4%, respectively for direct connections, Bright Data, Oxylabs, Proxyrack and Smartproxy. A negative value of δ_{RTT} occurs when the RTT$_{TCP}$ is higher than the RTT$_{TLS}$. This happens when the SYN-ACK and/or the ACK packets of the TCP connection are delayed but the subsequent packets in the TCP connections are not.

It can also happen that only the TLS packets are delayed while the initial TCP ones are not. In this case, the δ_{RTT} increases. This case is visually shown

[16] RESIP connections flagged as direct ones over the total amount of RESIP connections.

by the long tails of the distributions in Fig. 4.4. We note that, in the case of RESIP connections, the packets participating in the TLS handshake have a longer "journey" than the ones used to compute the RTT_{TCP}. Indeed, the packets travel from the client, through SUPERPROXY and GATEWAY, to the server. By contrast, in a direct connection, the packets only travel between GATEWAY and server. Hence, it is more likely to observe *an increase* of the δ_{RTT} caused by temporary network congestion rather than *a decrease*, for RESIP connections.

Fortunately, the above-mentioned situations, as reflected by the data of our experiment, are rare. To better understand how the variability of the network could influence our technique, we present a study of the variation of the RTT_{TCP} values per connection.

In our dataset, we have 45,902,917 connections for which the RTT of each TCP packet sent by the server and the corresponding ACK are recorded. These connections were generated in the last 56 days of the experiment. For each connection, we identify the minimum value of RTT_{TCP} and its position within the stream as well as the maximum RTT_{TCP} value and its corresponding position.

For our proxy detection technique to work, the ideal case is to have the minimum RTT_{TCP} in the first exchange and/or to have a low variability of its value throughout the connection. Our method could work taking as RTT_{TCP} a RTT value of later TCP exchanges (e.g. the minimum one of an entire connection). However, in this scenario, it would not enable us to detect the RESIP proxy at the very beginning of the connection (to possibly block it).

We find the minimum RTT in the first exchange in 56% of the connections and, generally, the variability is low (relative to our use case). More than half of the connections (53%) present a difference between the maximum and minimum RTT_{TCP} lower or equal to 50ms (our chosen threshold).

In 29% of the connections, the first RTT_{TCP} is the maximum one among all the exchanges. Even in these adversarial cases, the δ_{RTT} remains above the threshold for most connections, enabling our technique to work correctly.

Fig. 4.6 helps us to better visualize what happens to the δ_{RTT} when different RTT_{TCP} values are used to calculate the δ_{RTT}.

The graphs represent the CDFs of the δ_{RTT} (=RTT_{TLS}-RTT_{TCP}) when the RTT_{TCP} used to calculate it is the minimum (dashdotted lines) or the maximum one (full lines). The collection of the minimum and maximum values was performed in the second phase of the experiment when Bright Data had already stopped our subscription. Thus, it is not possible to represent the corresponding curves for this provider. We only show the δ_{RTT} values between 0 and 150ms for RESIP connections and the values between 0 and 60 ms for direct ones. In this way, we can better appreciate the distributions

Table 4.5: Analysis of the connections in which client, server, GATEWAY and SUPERPROXY are not further than 1000km from each other.

Provider	$\boldsymbol{\delta_{RTT} > 50}$**ms**	**Total Connections**
Bright Data	64.79%	22,582
Oxylabs	86.92%	12,887
Proxyrack	62.03%	79
Smartproxy	87.08%	14,314
Total	76.90%	49,862

near the 50ms threshold (dashed line).

As we can see in Fig. 4.6a, direct connections always present the large majority of connections below the threshold, no matter whether we use the maximum or the minimum value of the RTT$_{TCP}$. For the RESIP providers (Fig. 4.6b-Fig. 4.6d), we can also see that independently of the used RTT$_{TCP}$, most of the connections remain above the threshold. These figures, clearly show us that our technique remains valid, independently of the considered RTT$_{TCP}$.

Indeed, if we consider the very unlucky case where all first RTT$_{TCP}$ values would be the highest observed of that connection, we obtain a FPR of 0.01%, a FNR of 9.68%, and an accuracy value of 92.78%. Naturally, the percentage of false negatives increases with respect to our previous results (1.93%), but the accuracy remains high, even in this worst-case scenario.

When considering the minimum RTT$_{TCP}$, more than 99% of the connections present a positive δ_{RTT} for all providers and direct connections. When the maximum RTT$_{TCP}$ is considered, the percentage of negative δ_{RTT} increases to 12%, 6.9%, 5.8% and 6% for direct, Oxylabs, Proxyrack and Smartproxy connections respectively. The value for direct connections is surprisingly high. However, the majority of these negative δs$_{RTT}$ (65.92%) have a value of -1ms, showing that the RTT$_{TLS}$ and RTT$_{TCP}$ have still very similar values in this scenario.

The results of this section show that our technique is robust and can confidently detect RESIP connections even in very unlikely worst-case situations.

4.3.4 Machines Proximity Impact

Our detection technique is based on the correlation between the RTT and the distance traveled by packets. We assume that the RTT$_{TLS}$ is higher than the RTT$_{TCP}$ for RESIP connections because the TCP packets sent by the server stop at the GATEWAY. By contrast, the TLS ones, after reaching this machine, traverse the RESIP infrastructure and are then forwarded by the SUPERPROXY to the client.

However, we can consider a scenario in which client, server, GATEWAY and SUPERPROXY machines are in proximity to each other. In this situation, the overhead given by the physical distance of the machines taking part in the connection is small. Naturally, we could think that this influences the δ_{RTT} calculation and thus our detection.

For each connection, as explained in Section 4.2.4, we collect the GATEWAY and the SUPERPROXY IP addresses. Thanks to the MaxMind GeoLite2 database [70], we know the latitude and longitude of each of these IPs.

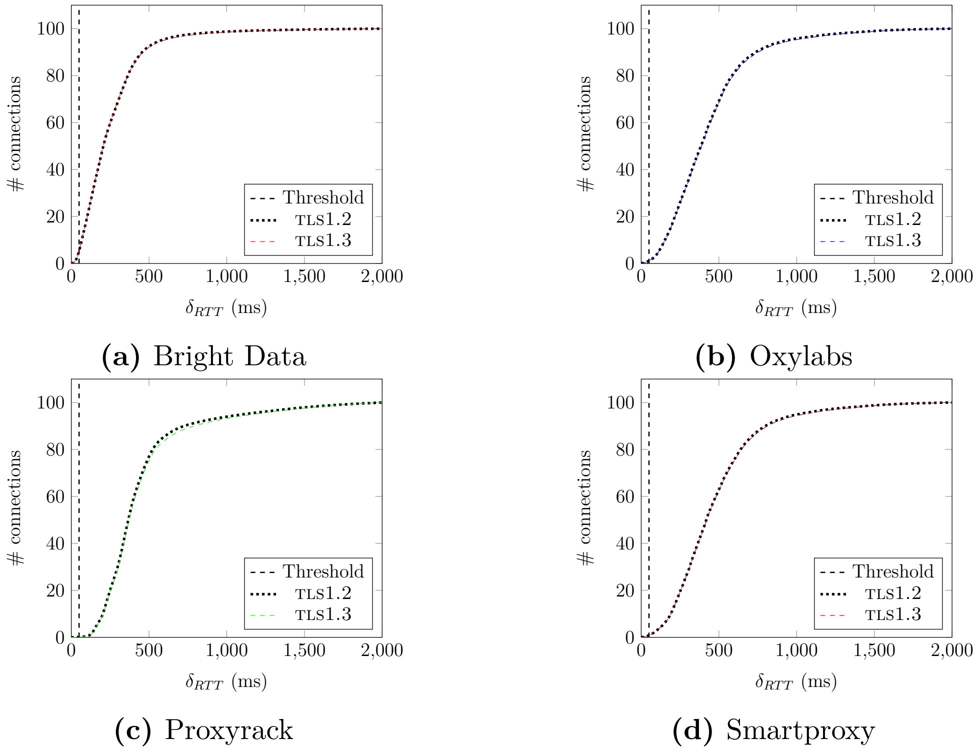
We consider those connections where client and server are in the same location and in which GATEWAY and SUPERPROXY are not further than 1000km (i.e. 10ms apart at a speed of 100km/s) from this location and from each other. We calculate the distances between the parties thanks to the Haversine formula [111].

Table. 4.5 shows the total number of connections satisfying the criteria and how many of them would correctly be labeled as RESIP, considering our 50ms threshold.

We can see that the total number of connections is low, especially for Proxyrack. In total, they account for 49,862 connections, which is 0.07% of the total amount of RESIP connections. This information tells us that, even when clients and servers are close to each other, it is not common that the SUPERPROXY is close to that location and/or the assigned GATEWAY is in near proximity to the other machines.

Considering all the providers together, we can see that still 76.90% of the considered connections have a δ_{RTT} higher than our chosen threshold (50ms). This tells us that, even in an unlikely event where all the machines are in near proximity, our technique still works relatively well. The exchanges between two additional machines (GATEWAY and SUPERPROXY) increase the δ_{RTT} enough for us to detect the presence of a RESIP infrastructure in more than 3 out 4 cases.

The number of false negatives increases with respect to the global data (23.09% against 1.93%). However, only 3.07% of these values present a δ_{RTT} lower than 20ms. This is the value under which we can find 97% of the δ_{RTT} of the direct connections (Section 4.3.2). Hence, this data shows that there is



(a) Bright Data

(b) Oxylabs

(c) Proxyrack

(d) Smartproxy

Figure 4.7: Comparison of the Cumulative Distribution Function of the δ_{RTT} for connections with TLS1.2 vs TLS1.3 (interval 0-2000ms).

still a significant difference between direct and RESIP connections.

4.3.5 The Impact of TLS Versions on the Threshold

In our experiment, we perform connections using TLS1.2 or TLS1.3. In both connections, we apply the same principle to perform our measurements, as described in Section 4.2.3. However, since the two protocols are different, different TLS packets are considered to calculate the RTT_{TLS}.

We could fear that these different choices in the considered packet could affect the registered δ_{RTT}. For example, the processing at the client side to generate the packet P2 could be longer/shorter for one TLS protocol than the other. In this case, two servers running different TLS versions and implementing our detection method with the same threshold would produce different accuracy levels of the technique.

In this section, we analyze if the above-mentioned scenario takes place and thus if the TLS version run by the server impacts our technique. We have studied separately the communications performed using TLS1.2 and TLS1.3.

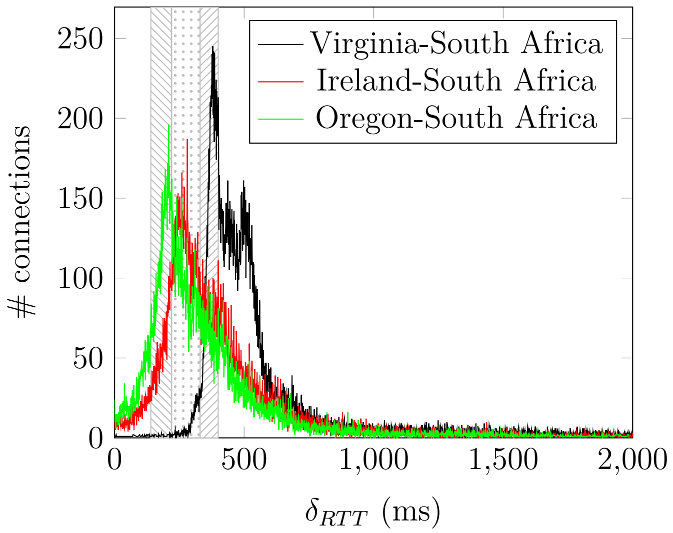


Figure 4.8: δ_{RTT} values distribution for three representative client-server paths (Proxyrack).

Fig. 4.7a-Fig. 4.7d show, with two different colors and patterns, the δs_{RTT} cumulative distribution functions for TLS1.2 and TLS1.3 for each provider. We have maintained the same x-axis ranges used in Fig, 4.5. The shapes of both curves overlap and they are in line with the global distributions. These results clearly show that the TLS version run by the server does not impact our measurement.

4.3.6 δ_{RTT} Distribution Shapes

The attentive reader can see that different providers show distinctive shapes in the δ_{RTT} values distributions (Fig. 4.4). Bright Data distribution has peaks in the intervals 50ms-210ms and 250ms-370ms, identified with different backgrounds in Fig 4.4a. Fig. 4.4b and Fig. 4.4d show Oxylabs and Smartproxy distributions of δ_{RTT}. The curves present similar Gaussian shapes with oscillations. The curves present similar shapes. In Fig. 4.4c we can see the distribution of the δ_{RTT} in Proxyrack connections. It displays peaks (areas with different backgrounds) in the intervals 140ms-220, 220ms-330 and 330ms-400ms.

The cause of these peculiar shapes can be identified by considering the relative geographic position of the client and the server involved in each connection.

For each RESIP provider, we define S as the set of all collected connections and S_{ij} the subset of connections sent to server j by client i. For each S_{ij}, we examine the distribution of the δs_{RTT}. The results show that the shapes of the distributions differ from the ones for the whole set S, shown in Fig. 4.4. In particular, each distribution shows a peak of connections in a specific small range of δ_{RTT} values and low values for the rest of the x-axis. Studying the individual shapes and their composition, we can conclude that the peaks are created by the overlap of distributions with distinctive shapes shifted on the x-axis.

Fig. 4.8 shows representative δs_{RTT} distributions of three S_{ij} from Proxyrack. Each of them has a peak in a different interval among the [140-220]ms, [220-330]ms, and [330-400]ms in which we previously noted the peaks (areas with different backgrounds in Fig. 4.4c). The not shown S_{ij} present δs_{RTT} distributions with shapes and values similar to those in the figure. Since $\sum_{i,j} S_{ij} = S$, Fig. 4.4c is the sum of the δ_{RTT} distributions of each Sij, which have the shapes provided in Fig. 4.8. The peaks of Fig. 4.4c are the result of the superposition of the distributions of each S_{ij}, which have similar shapes but are shifted along the x-axis. For the distributions of the other providers, the same schema takes place.

We need to remember that the RTT$_{TLS}$ is a measure of the distance client-server and the RTT$_{TCP}$ is a measure of the distance GATEWAY-server, in the case of RESIP connections. Thus, the δ_{RTT}, which is the difference between RTT$_{TLS}$ and RTT$_{TCP}$, gives us a measure of the distance between client and SUPERPROXY, plus the RESIP overhead and the distance between SUPERPROXY and GATEWAY. The differences in the δ_{RTT} distributions of the S_{ij} suggest that the distances client-SUPERPROXY and SUPERPROXY-GATEWAY are mostly influenced by the client location, instead of the server one. Indeed, studying the distributions of each S_j, the subset of connections sent to server j by all clients, we find shapes similar to the ones in Fig. 4.4 and we do not observe differences among them, contrary to what we see considering the S_{ij}.

Thus, the bigger the shift on the x-axis of a peak of an S_{ij} is, the bigger the distances among client, SUPERPROXY and GATEWAY likely are. We believe this happens when there are no available GATEWAY and/or SUPERPROXY in the proximity of the client.

4.3.7 Ground Truth Experiment Discussion

In the previous subsections, we have investigated different aspects of the RTT_DETECTION technique thanks to the connections in RTT_DS. Hereafter, we discuss the results of our analyses in terms of the validity and feasibility of the technique.

Thanks to our analyses, we can see that the δ_{RTT} values of RESIP connections are substantially higher than the ones of direct connections. 97.05% of the direct connections have an δ_{RTT} lower than 50ms, our chosen threshold. In the same interval of δ_{RTT} ([0,50]ms), Bright Data, Oxylabs, Proxyrack and Smartproxy connections account for, respectively, the 5.35%, 1.08%, 0.13% and 0.81% of the total. The accuracy of our method remains high even in conditions of network delays (Section 4.3.3). This happens even when client, server and SUPERPROXY and GATEWAYs are in close proximity (Section 4.3.4). Different TLS versions on the server do not impact our measurement (Section 4.3.5). This data shows that the technique is robust in differentiating RESIP and direct connections from the same device.

The RTT_DETECTION technique can be easily implemented on existing servers, without modifying existing software. Indeed, we can perform the measurement outside the server with a sniffer program, as we do in our setup. To mitigate identified RESIP connections, the sniffer program can signal the result of its analyses to a mitigation software of choice.

To efficiently communicate the detection verdict to another software, the sniffer needs to promptly capture and analyze the TCP and TLS packets. For this reason, the machines running the RTT_DETECTION technique need a memory capacity that enables this operation on top of the one needed to run the server program. The memory consumption of the sniffer depends on the software used to implement it and the average number of concurrent requests that the server program can handle.

Finally, we can apply the RTT_DETECTION in the majority of Internet transactions. The technique works for HTTPS connections and, nowadays, more than 79% of websites use secure encryption. Moreover, this percentage grows every year [112]. Furthermore, as we saw in Section 4.3.5, the technique can work with both TLS1.2 and TLS1.3 servers. This information tells us that it is feasible to implement the technique to protect most web domains.

4.4 Client Environment Analysis

In the previous sections, we presented the Ground Truth Experiment and the results we collected running it. Thanks to this experiment, we were able to assess that the δ_{RTT} is significantly higher for RESIP connections than direct ones starting from the same device. In this section, we focus on direct connections and the possibility that the client environment in which they are generated impacts our detection.

Our technique measures the RTTs at the server side. Hence, beyond reflecting the distance traveled by packets, the RTT$_{TLS}$ includes the client application layer processing time. In the Ground Truth Experiment setup, we performed connections among well-connected data center machines using Python scripts. This is not a common way for end users to connect to websites. Generally, they utilize personal devices, such as laptops and mobile phones. They navigate the Internet thanks to web browsers and can leverage hotspots. In Section 4.4.1, we provide evidence that these factors do not impact our measurement.

Moreover, the RTT_DETECTION technique does not recognize only RESIP connections. It identifies all proxies that break the TCP session while keeping the TLS end-to-end between these two parties. RESIPs are not the only proxies behaving in this way.

An example of this is SSH forwarding. Both local and remote forwardings do not provide end-to-end TCP connections and are thus detected with our method. Specific VPNs (e.g. WireGuard protocol [113]) and Tor [114] behave in the same way and are thus detected as RESIP by our technique.

Furthermore, Mobile TCP Terminating Proxies (MTTPS) are another example of proxies that break the TCP but not the TLS session [95]. MTTPS belong to specific mobile Internet Service Providers (ISPs) and break the TCP session between the device and a server at the antenna. This enhances performance since the probability of packet loss is higher in the electromagnetic wave transmission between the device and the antenna than in the wired part of the connection. Breaking the TCP, the device needs to resend lost packets only for the short path between the antenna and the device itself.

On the other hand, there are network artifacts that do not behave like RESIP by design. A widely deployed defense for companies is the Web Application Firewall (WAF). WAFS need to break TCP and TLS to study the application content and assess if the communication is allowed to continue. Thus, if a client is behind a WAF, both TCP and TLS between client and server are broken and we see comparable RTT$_{TCP}$ and RTT$_{TLS}$ measurements. The RTT_DETECTION technique does not declare these connections as RESIP.

Secure tunneling solutions using Network Address Translation (NAT) and IPSEC do not break TCP, but leverage encapsulation. This technique is nowadays widely used both in private and commercial networks. Since they do not break the TCP connection, the RTT_DETECTION technique does not classify them as RESIP requests.

We need to reach a better understanding of how much delay non RESIP detected proxies add to the δ_{RTT} with respect to the direct connections and RESIP ones. In this way, we can understand if we can discern between the connections coming from these proxies and the ones from RESIP and tune our threshold accordingly to limit false positives. For this reason, we test direct connections with varying client-side environment conditions (Section 4.4.1).

While it is crucial to understand how various factors impact the δ_{RTT}, it is also important to understand the probability of having these factors in the connections. In Section 4.4.2, we take advantage of real-world connections to assess this probability. Our preliminary results suggest that the usage of mobile ISPs that leverage MTTPS causes on average one-third of the false positives. Moreover, in those cases, the additional delay in the δ_{RTT} is smaller than the one provoked by the RESIP infrastructures and other identified TCP terminating proxies. This opens the door to the possibility of discerning when each of the two contributions increases the δ_{RTT}.

4.4.1 Preliminary Client Environment Analysis

In this section, we carry out a preliminary analysis of client-side factors and their impact on the RTT_DETECTION technique.

Experimental Setup

A large number of devices, types of networks, web browsers and tunneling techniques exist nowadays. It is not feasible to consider all the available combinations. For this reason, we consider this experiment to be explorative and not exhaustive. We chose an initial subset of the available combinations, based on the possibilities we had at the time we performed the test. We consider this a starting point in our analysis. Furthermore, the approach we adopt is cumulative, thus, it is possible to test other combinations in the future and evaluate the results with the ones obtained thanks to this first initial sample. We leave the expansion of this analysis as a future work for this thesis.

For this experiment, we use a personal computer and a mobile phone as client machines. With these devices, located in France, we query one of the servers of the Ground Truth Experiment (Section 4.2.2) in the United Arab Emirates. The server runs the RTT_DETECTION technique.

We perform the connections with different configurations. For each configuration, we send 20 connections to the server which is running TLS1.2. As explained in Section 4.3.5, the TLS version does not impact the measurement hence this choice does not bias the results.

With the personal computer, we connect to the Internet leveraging three different setups. In the first one, the personal computer reaches the Internet thanks to a 5GHz Wi-Fi signal from a residential access point. In the other setups, we take advantage of the mobile device included in the experiment and we use it as a hotspot. The personal computer has access to the Internet thanks to this. First, the mobile phone is connected to the 5GHz Wi-Fi signal presented above, then to the 4G network thanks to the French Provider SFR

[115]. In this way, we distinguish the contributions on the δ_{RTT} of the wired and the SFR mobile networks as well as the impact of performing hotspots.

We test the above-mentioned settings using no VPNs, Tor [114] and NordVPN [116]. We use it with the WireGuard protocol and we leverage their desktop application to establish the VPN. We use these connections to benchmark the contributions of detected non RESIP connections. For these connections, we randomly choose a new exit location for each request.

We perform connections using two different browsers, Google Chrome [117] and Microsoft Edge [118]. These browsers are among the most used by genuine users [119]. For the Tor connections, we use the Tor Browser. We manually perform the queries to the server, to reproduce a real user interaction. We also send queries with Python [101] scripts. We use them as a benchmark to assess the delays introduced by web browsers.

For the mobile device, we test two types of networks: Wi-Fi and 4G (from SFR). We query the server using no VPN and the Google Chrome Browser. We perform these connections to evaluate the delay of the mobile network without the hotspot contribution.

Results

Table 4.6 provides a summary of all the tested combinations and the results of the measurement. We present the median value plus or minus the median absolute deviation of the twenty collected δ_{RTT} values. Hereafter, we use the different colors of the cells to help read the table and thus interpret the results.

When examining the combinations where no VPN is used and connections are made via Wi-Fi (light blue and light green cells), we observe that the δ_{RTT} remains below the threshold of 50ms for both devices, even when using the Wi-Fi hotspot. However, in the cases where connections are made through 4G directly from the mobile device or via the hotspot and without using a VPN (dark blue and dark green cells), the δ_{RTT} exceeds 50ms. This discrepancy suggests that SFR (the mobile ISP used in this experiment) utilizes TCP terminating proxies, which contribute to the higher δ_{RTT} values observed in these scenarios.

As expected, NordVPN and Tor are detected as a RESIP when using Wi-Fi (light gray and light orange cells). When connecting with NordVPN and Tor through 4G (dark gray and dark orange cells), the δ_{RTT} is generally higher compared to when using the Wi-Fi (respectively light gray and light orange cells). These findings indicate that NordVPN, Tor, and the SFR mobile network itself each contribute to a delay independently. In cases where a connection involves two of these factors simultaneously, the delays associated with each

Table 4.6: Tested combinations to assess the impact of different factors on false positives with the respective median value plus or minus the median absolute deviation of the δ_{RTT}.

Device	**VPN**	**Network**	**Web Browser**	$\boldsymbol{\delta_{RTT}}$ **(ms)**
Personal computer	No VPN	Wi-Fi	Google Chrome	28±16.5
			Microsoft Edge	13±6.5
			Python	3±1
		Hotspot (Wi-Fi)	Google Chrome	36±10
			Microsoft Edge	32±15.5
			Python	3.5±2.5
		Hotspot (4G)	Google Chrome	117±56
			Microsoft Edge	77±18
			Python	52.5±6.5
	NordVPN	Wi-Fi	Google Chrome	127±4.5
			Microsoft Edge	158±14.5
			Python	140±20
		Hotspot (Wi-Fi)	Google Chrome	127.5±1.5
			Microsoft Edge	129±1.5
			Python	124.5±5
		Hotspot (4G)	Google Chrome	188±34
			Microsoft Edge	170±8
			Python	219.5±70.5
	Tor	Wi-Fi	Tor Browser	278±50.5
		Hotspot (Wi-Fi)	Tor Browser	982.5±274
		Hotspot (4G)	Tor Browser	888±183
Mobile phone	No VPN	Wi-Fi	Google Chrome	10.5±6
		4G	Google Chrome	51.5±14.5

factor are combined, resulting in a bigger δ_{RTT}.

We can observe that using a web browser introduces a delay compared to using a Python script. This is evident in the tested combinations involving browsers and Python (blue and gray cells). However, the median values of the δ_{RTT} remain below the threshold when no other high-delaying factors are present, such as 4G, NordVPN, and Tor (light blue cells). This demonstrates that the use of web browsers alone does not lead to false positives.

Furthermore, when using a hotspot to connect to a Wi-Fi signal, there is an increase in the δ_{RTT} compared to a direct Wi-Fi connection (light blue, light gray, light orange cells). However, neither this factor alone results in δ_{RTT} values exceeding 50ms.

In addition to 4G connections, Tor and NordVPN are the other significant sources of false positives (gray and orange cells). Currently, the RTT_DETECTION technique identifies these connections as RESIP. To distinguish their respective contributions (RESIP and VPN/Tor), one approach is leveraging the knowledge of the IP addresses of the exit nodes of these parties. We can reach this knowledge by taking advantage of commercial databases that provide this information (e.g. *IPQualityScore* [120]) and directly checking the list of exit nodes for Tor. Another potential approach consists of performing Maximum Transmission Unit (MTU) analysis at the server side, expanding upon prior studies to detect VPNs [51], [121]. We leave this differentiation as a future work after this thesis.

When considering only 4G as the delaying factor (dark green cell), we observe that the δ_{RTT} exceeds the threshold. However, the delay introduced solely by 4G is much smaller compared to the delays introduced by NordVPN and Tor when considered independently (light gray and light orange cells).

As previously discussed, ISPs employ MTTPS to enhance performance. Our findings support this, as the delay caused by the 4G provided through SFR is lower than in other cases. Moreover, the average value of 51.5±14.5ms is close to the threshold. This suggests that, if this finding applies to other ISPs using TCP terminating proxies, we could potentially mitigate false positives caused by this setup by adjusting the threshold. Hereafter, we further explore this hypothesis by considering real-world connections.

4.4.2 Detection on Real-World Scrapers Connections

In the previous section, we conducted an initial analysis of the client environment factors that can impact the RTT_DETECTION technique. In this section, we leverage real-world connections to assess the probability of encountering such factors. Additionally, we explore the extent to which Mobile TCP Terminating Proxies (MTTPS) contribute to this probability. Building upon the results we obtained in the previous analysis, we explore the feasibility of distinguishing between the contributions in the δ_{RTT} of RESIPs and these proxies.

The positive results of the Ground Truth Experiment helped us in convincing Amadeus IT Group (Amadeus) to adopt the RTT_DETECTION technique. As described in Section 2.2.3, Amadeus offers various IT solutions to airlines, including products that enable passengers to make flight bookings. Web scraping heavily impacts these solutions. Amadeus employs a third-party Bot Detector to protect their systems. The Bot Detector intercepts each request directed to the Amadeus booking domains and scrutinizes the request. If a request matches the parameters of a bot signature, the Bot Detector responds with a countermeasure. Otherwise, the request proceeds to the intended booking domain.

The RTT_DETECTION technique is based on measuring the RTTs among client, GATEWAY and server. When a domain is protected by the Bot Detector, this party acts as the server. Consequently, it becomes necessary for us to conduct measurements on that end. We successfully convinced the third-party company to integrate the RTT_DETECTION technique into their Bot Detector. This collaboration enables us to examine the effects of our technique on the connections that reach Amadeus domains.

The Bot Detector registers the RTT_{TCP} and the RTT_{TLS} values for each newly established connection. Amadeus has the option to utilize the δ_{RTT}, either independently or in conjunction with other indicators supplied by the third-party company, to construct custom rules for specific pathways.

Experience shows that, by employing the RTT_DETECTION technique to protect Amadeus domains, the δ_{RTT} serves as a robust parameter for identifying connections that traverse RESIPs. Over the course of two representative months, the δ_{RTT} was incorporated as a custom rule parameter in 74.32% of the investigations. Indeed, during this period, a total of 148 rules were established for domains targeted by scraping campaigns, with 110 of them including the δ_{RTT} as a parameter.

As discussed in Section 4.4.1, various factors (specific VPNs, Tor, MTTPS) lead to false positives. Consequently, we do not deem it reliable to rely solely on δ_{RTT} as a standalone parameter at this stage. It is crucial to determine the percentage of traffic in which different factors contribute to false positives. This understanding will aid in assessing their impact and significance in generating false positives. In the following investigation, we utilize the real-world connections reaching Amadeus to evaluate this probability. Furthermore, we explore the extent to which we can attribute false positives to MTTPS.

False Positive Ratio in Airlines Booking Domains

On the airline domains managed by Amadeus, users reach a designated page (*Booked Flight Page*) after they have successfully made a payment for their booking. Our assumption is that only genuine users who have purchased flight tickets will reach this page. As a result, we consider any connection that reaches the *Booked Flight Page* page with an δ_{RTT} equal to or greater than our threshold value of 50ms as a false positive connection[17].

For a representative week (01/06/23-07/06/23), we collected the IP addresses and the corresponding δ_{RTT} of the connections reaching the *Booked Flight Page* of eight different airlines. We categorize the airlines by size. For a Big airline, on average more than 3,000 IPs reach the *Booked Flight Page* page daily. For a Medium-Big one, the average is between 1,500 and 3,000 while, for a Medium-Small one, this value is between 100 and 1,500. We consider Small those airlines whose average number of IPs is lower than 100 per day. For each airline, we calculate the mean and standard deviation of the daily percentage of false positive IPs, the daily False Positive Ratio (FPR). The third column of Table 4.7 shows the results of this calculation.

The average among all airlines is set at 17.66%. However, the standard deviation is high (7.53%). This tells us that the percentage of false positives largely varies among the different airlines. Indeed, some airlines have a much higher false positive ratio (*Airline* E, 29.27%), while others show much smaller ones (*Airline* H, 7.24%). Furthermore, there is not a direct correlation between the mean and/or standard deviation values and the size of the airline. The mean value varies among the airlines of the same category. Moreover, while we could expect the standard deviation to increase with the decrease in the size of the airline (a lower number of total IPs is more affected by even small changes), we see that this is not always the case. The airlines in the category Big do not have the lowest standard deviation among all categories.

The reported percentages are considerably higher than what we observed in the Ground Truth Experiment. This indicates that the probability of finding a δ_{RTT} delaying factor is high in real-world connections.

According to [122], in 2022, 40% of the United States people used a VPN and the majority of them did so for personal reasons. As previously discussed (Section 4.4.1), specific VPNs are detected as RESIP by our technique. Thus,

[17] Although the assumption we make is reasonable, it is worth acknowledging that there could be rare instances where RESIP campaigns aim to purchase tickets legitimately. While we recognize the possibility of such events, we currently lack the means to distinguish this particular scenario from other situations that generate genuine false positives. Therefore, it is important to treat the positive percentages presented in this section as an upper-bound estimate for the number of false positives associated with the technique.

Table 4.7: False positive ratio and mobile δ_{RTT} of analyzed airlines.

Airline	Size	FPR (%)	Mobile FPR (%)	Non Mobile FPR (%)	Mobile FPR δ_{RTT} (ms)
Airline A	Big	17.86±1.49	34.00±3.48	66.00±3.48	90±8
Airline B	Big	10.54±1.07	24.04±10.32	75.96±10.32	82±1
Airline D	Medium-Big	25.65±0.76	60.26±2.73	39.74±2.73	72±2
Airline C	Medium-Big	12.51±0.63	36.08±2.25	63.92±2.25	147.5±12.5
Airline E	Medium-Small	29.27±1.36	58.74±4.05	41.26±4.05	90±7
Airline F	Medium-Small	22.15±1.87	37.80±6.70	62.20±6.70	93±6
Airline H	Small	7.24±3.10	0.00±0.00	100.00±0.00	NA[18]
Airline I	Small	16.10±5.10	3.09±5.90	96.91±5.90	NA[18]
All airlines	-	17.66±7.53	31.75±21.66	68.25±21.66	88.5±59.0

our intuition is that VPNs are one of the major causes of the false positives in this scenario.

Moreover, we know that we detect Mobile TCP Terminating Proxies (MTTPS) as RESIP. Although the exact percentage of mobile ISP utilizing these proxies is unknown, we know that nowadays a significant portion of connections is made through mobile networks. Moreover, this trend is expected to continue growing in the coming years [123]. Given this context, it is reasonable to assume that TCP terminating proxies play a substantial role in generating false positives in such scenarios.

In Section 4.4.1, we saw that SFR mobile connections showed an δ_{RTT} close to the threshold. Hereafter, we show that also real-world mobile connections that result in false positives show a similar trend. Thanks to this, we could differentiate TCP terminating proxies and RESIPs and reduce the probability of false positives.

Mobile Connections δ_{RTT}

To gain further insights in studying the δ_{RTT} of mobile connections, we leverage the Mobile Carrier Database of Digital Element [124]. We use it to

[18] Not Applicable to those airlines in which the number of false positive connections is lower than one per day.

categorize the number of false positive mobile connections[19] among the ones previously considered. In Table 4.7, the third and fourth columns indicate the daily distribution of the FPR between mobile and non-mobile connections.

We can see that the mobile FPR varies greatly among different airlines. Mobile connections do not cause any false positive in *Airline* H whereas they are a significant factor for both *Airline* D and *Airline* E. However, we observe that globally the FPR reduces to 10.13±3.54 if we consider only non-mobile FPR. This means that if we manage to discern between the mobile and RESIP contributions in the δ_{RTT}, we can lower the average FPR of all airlines by one-third.

The last column of Table 4.7 proposes the median value and the median absolute deviation of the δ_{RTT} when a mobile connection results in a false positive. With the elements at our disposal, we can not know with certainty that the only factor that increases the δ_{RTT} in these connections is the usage of the mobile network. There could be other factors, on top of it, that increase the delay. However, based on the results in Section 4.4.1, the reported delays are generally[20] more compatible to the connections using SFR 4G with a web browser and/or hotpot than the ones leveraging NordVPN or Tor. This seems to suggest that the main contributor to the delay is the usage of mobile networks.

In the case of mobile connections FPRs, the δ_{RTT} is higher than the threshold but still lower than the majority of values presented by RESIP connections. If we consider the highest median δ_{RTT} among those values plus the corresponding median absolute deviation we obtain 160ms (*Airline* C). In the Ground Truth Experiment, 90.63% of RESIP connections have a δ_{RTT} higher than this value. These results suggest that it could be possible to recognize false positives produced by recognized mobile connections thanks to the lower δ_{RTT} they expose. To bring more evidence to this point, we analyze true positive connections. Hereafter, we describe the collection and analysis of these connections.

True Positive Connections Case Study

As previously described, Amadeus employs a custom rule-based Bot Detector to identify and mitigate scraping requests. In order to assess the impact of mobile connections on the δ_{RTT}, we need a custom rule in which the δ_{RTT} was not a parameter of detection but where at the same time the vast majority of

[19] In the rest of this section, we use mobile connections to define those connections originating from mobile networks.

[20] All airlines for which it was possible to calculate the Mobile FPR δ_{RTT} except *Airline* C.

the connections had an δ_{RTT} greater than 50ms.

We discovered a suitable rule on *Airline* X that fulfilled our requirements. Among the IPs that matched this rule, less than 2% had an δ_{RTT} below the specified threshold.

We analyzed the connections that matched the rule and their RTT_DETECTION on a specific day (12/06/23). Leveraging the Mobile Carrier Database of Digital Element, we classified the IPs into mobile and non-mobile categories.

Our analysis reveals that 10.93% of the IPs belong to mobile connections. While the median δ_{RTT} for non-mobile connections is 251ms, for mobile connections it is 313ms, indicating a difference of 62ms. This finding supports the fact that mobile networks (using MTTPS) introduce a delay in the δ_{RTT} (62ms) that is lower than that of RESIPs (251ms). Moreover, when the two factors take place in the same connection, the delay increases. Consequently, it may be possible to use a higher value for the threshold of the RTT_DETECTION for mobile connections to not detect them as false positives. A comprehensive sensitivity analysis is required to determine the most suitable value for this adjustment.

4.5 Summary

In this chapter, we conducted an evaluation of the RTT_DETECTION technique to assess its validity and feasibility. The technique classifies connections as originating from RESIP when their RTT$_{TLS}$ exceeds the RTT$_{TCP}$.

Through our experiments, we consistently observed that the δ_{RTT} (calculated as RTT$_{TLS}$- RTT$_{TCP}$) exhibited significantly higher values for RESIP connections compared to direct connections when the same client was utilized in both scenarios. The initial results indicate that a threshold of 50ms is the most effective in distinguishing between direct connections and RESIP connections from the same device.

Our analyses of specific client environments reveal that the technique can encounter false positives when a mobile network utilizes Mobile TCP Terminating Proxies or when requests are routed through specific VPNs and Tor. Real-world connection studies demonstrated a relatively high percentage of false positives (17.66±7.53). This makes the technique unsuitable for standalone use. Nevertheless, a real-world company currently uses the technique in combination with other parameters to detect scrapers with successful results. This shows that a high δ_{RTT} is a strong indicator of RESIP activity and false positives can be reduced by using this detection in conjunction with other parameters.

Furthermore, preliminary analyses suggest that by increasing the threshold, it may be possible to mitigate false positives caused by mobile networks employing Mobile TCP Terminating Proxies. This adjustment could potentially reduce the false positive rate by one-third.

As explained in Chapter 3, RTT_DETECTION is just one of the two techniques that we identified to detect RESIP connections at the server side. In the next chapter, we present the measurement campaign we performed to validate our second technique, MRP_DETECTION. This second technique enables us to detect RESIP connections originating from specific providers. Thus, on top of detection, it provides attribution. In the next chapter, after presenting the validation of the technique, we propose a possible combination with RTT_DETECTION to differentiate the contributions of the different RESIP providers and reduce the possible user degradation caused by the MRP_DETECTION technique standalone.

Chapter 5

Scraping Detection through Retransmission Protocols

5.1 Introduction

As introduced in Section 3.5, when a client uses a RESIP to send a SYN packet to a non-acknowledging server, we witness a behavior different from the one of direct connections. If no RESIP is involved, only one IP address contacts the server and the corresponding machine simply keeps retransmitting the original SYN packet from the same port with the exponential backoff behavior associated with the retransmission timeout.

On the other hand, when the client hides behind a RESIP, new GATEWAYs contact the server. Moreover, GATEWAYs can act according to one of two Machine Retransmission Protocols (MRPs). In the first MRP, the GATEWAY retransmits the original SYN packet, as when no RESIP is involved. In the second MRP, the GATEWAY closes the connection at the application layer and opens a new one from another port. In this case, the server receives a new SYN packet. Moreover, if the application timeout is longer than the kernel one, the GATEWAY retransmits also once the original SYN packet.

In theory, one could think of exploiting the usage of the behavior of the second MRP to detect, server-side, when a connection passes through a RESIP service (MRP_DETECTION).

The server can delay sending SYN-ACK packets and check, for each client, if it sends a new SYN packet when the original one is not acknowledged in time. This identifies a RESIP connection.

In this chapter, we present the measurement campaign we performed to characterize three RESIP providers when contacting a non-acknowledging server. Our goal is to check whether the specific behavior seen in our preliminary analysis holds over time and assess the validity and feasibility of the MRP_DETECTION technique.

Hereafter, we present the setup used to perform our measurement campaign (Section 5.2). Then, Section 5.3 shows the analyses performed on the collected communications. Finally, Section 5.4 discusses the MRP_DETECTION technique and its feasibility.

The information provided in this chapter, as well as the explanation of the idea underlying the MRP_DETECTION, were published in [C5].

5.2 Experimental Setup

In our experiment, we consider two machines, SERVER_A and SERVER_B. These machines are located in Ireland and supplied by the same provider (Azure [100]). Contrary to the RTT_DETECTION technique (Section 3.4), preliminary analysis showed that the geolocalization of the clients and servers does not influence the behavior of RESIP when contacting a non-acknowledging server. Thus, for this experiment, we selected only one location for our machines among the available ones.

SERVER_A and SERVER_B are reachable from the public Internet on port 80 but they do not reply to SYN packets. They do not generate ICMP error messages either for connections to this port. In this setup, SERVER_A tries to send HTTP GET requests to SERVER_B through a RESIP provider. Every time the client sends a request, it waits until the SUPERPROXY closes the connection. After that, it sends a request using another RESIP provider tested with this setup. On the contrary, we do not send any requests to SERVER_A. Thus, SERVER_A only receives SYN packets from external sources, if any.

When SERVER_B receives a SYN packet, it does not have information about the real initiator of the request (the machine behind the RESIP). Indeed, for RTT_DETECTION, we encoded this information in the URL (Section 4.2). In this new setup, we do not have this possibility since we send just one SYN packet. Moreover, SERVER_A does not have any knowledge about which GATEWAY IPs the SUPERPROXY assigns to its outgoing requests. Hence, there is no direct way to match the requests sent by SERVER_A with the ones received by SERVER_B. Furthermore, SERVER_B is publicly reachable and scanning campaigns most likely produce some of the SYN packets sent to it. To keep only the connections originating from RESIP GATEWAYs that reach SERVER_B, we can use the traffic that reaches SERVER_A as a reference. SERVER_A and SERVER_B share the exact same location and provider. Hence, we can assume that they are frequently scanned at the same time by the same campaigns or, at least, that they witness such scans at a similar rate.

Table 5.1: Attributes of a communication in the UNACKED_DS dataset.

Attribute	Attribute explanation
COM_START_EPOCH	Epoch (UTC+0) in which the client sends the request and thus starts the communication [s]
COM_END_EPOCH	Epoch (UTC+0) in which the client receives the signal by the SUPERPROXY to end the connection [s]
N_IPS	Number of different IP addresses/GATEWAYs that send SYN packets during the communication
AVG_SYN_PER_IP	Average number of SYN packets an IP sends in the communication
AVG_SYN_RET_PER_IP	Average number of retransmissions per IP performed in the communication
AVG_DIST_COUNTRIES	Average number of distinct countries associated with the IPs of the communication
AVG_DIST_CONTINENTS	Average number of distinct continents associated with the IPs of the communication
MEDIAN_NEW_SYN	Median delay between SYN packets the same IP issues from different ports [s]
INT_DIFF_IPs	Elapsed intervals between a request from an IP and the next request with a new IP in the communication
PERC_PARALLEL_IPs	Percentage of IP addresses that send packets in parallel
BEHAVIORS_COUNTS	Counters for each type of behavior of an IP: only retransmissions, only sending a new SYN from a new port, doing both actions

For 88 days (03/02/2022 at 09:00 UTC +0 to 01/05/2022 at 09:00 UTC +0) we recorded connections to SERVER_A and SERVER_B. We queried SERVER_B using three RESIP providers: Oxylabs (OL) [92], Proxyrack (PR) [93], and Smartproxy (SM) [94]. In total, we recorded 9,219 incoming connections to SERVER_A and 1,773,407 incoming connections to SERVER_B.

As explained above, we can see the connections to SERVER_A as a reference for the scanning activity on SERVER_B. We can then use them to clean the connections to SERVER_B and keep only the ones produced by the GATEWAYs. For each connection performed to SERVER_B, if the same IP address contacted SERVER_A in the same hour, we eliminate the entries associated with that IP address and time from the logs of SERVER_B and SERVER_A. These are very likely SYN packets generated by scanners, not by a RESIP. Thanks to this operation, we can delete 1,666 connections from SERVER_B logs (0.09% of SERVER_B connections) and 1,840 connections from SERVER_A ones (19.96% of SERVER_A connections).

Moreover, we can consider the number of connections per hour received by SERVER_A and by SERVER_B in these cleaned datasets. If the two values were similar, it would mean that the amount of scanning activities is comparable to the number of requests sent by GATEWAYs and thus it would be impossible for us to distinguish the two contributions.

On average, the amount of requests received at SERVER_A corresponds to 0.43% of the amount received by SERVER_B. This confirms that RESIP GATEWAYs, and not scanners, produced the vast majority of connections received by SERVER_B. Even if a small number of requests at SERVER_B are scanning ones, they are lost in the noise of the requests generated by SERVER_A and we can safely ignore them for the statistical analysis we intend to do. Thus, this clean dataset enables us to have a good representation of the behavior of RESIP GATEWAYs when the TCP connection can be initiated but not completed.

For each connection initiated by our client, there are many SYN packets received at the server from different GATEWAYs. We match each connection started at SERVER_A with all the connections at SERVER_B that were received between when SERVER_A started the request and when the SUPERPROXY ended the request. We define this as a communication. In total, this dataset, which we call UNACKED_DS, contains 124,865 communications. This value corresponds to the number of connections initiated by SERVER_A. On SERVER_B, it corresponds to 1,734,351 incoming SYN packets. Table 5.1 shows the attributes we consider for each communication in the UNACKED_DS and the corresponding explanation. In the next section, we present the analyses performed on the dataset and discuss their results.

5.3 Communications Analysis

In this section, we analyze the communications in UNACKED_DS and we share the insights gathered from these analyses. Table 5.2 shows statistics about

Table 5.2: Mean value and standard deviation of attributes (UNACKED_DS).

RESIP	Duration (s)	N_IPS	AVG_SYN_PER_IP	AVG_SYN_RET_PER_IP
OL	52.39±97.02	3.18±0.76	3.07±0.85	1.8±1.14
PR	15.77±5.19	3.27±0.78	1.04±0.15	2.12±0.5
SM	51.87±96.33	3.98±0.87	2.66±0.7	1.66±0.88

Table 5.3: Mean value and standard deviation of the average number of distinct countries and continents in UNACKED_DS.

RESIP	AVG_DIST_COUNTRIES	AVG_DIST_CONTINENTS
OL	2.87±0.78	2.24±0.71
PR	3.08±0.82	2.15±0.75
SM	3.61±0.87	2.62±0.77

the attributes of the communications. It presents for each attribute the average plus or minus the standard deviation. The second column tells the duration, registered at the client, of a communication (COM_END_EPOCH-COM_START_EPOCH). This value informs about the timeout set in the RESIP to determine when to abandon a connection. For the RESIP provider PR, we have a low value and small variability. This tells us that most likely this interval is fixed for this provider. On the other hand OL and SM show higher values and higher variability. This is more compatible with a not fixed interval. Let us now consider if the multiple used GATEWAYs appear in sequence or partially in parallel. The median percentage of a new IP appearing while the previous one is still sending packets (PERC_PARALLEL_IPS) is 0% for OL and SM. This value is 50% for PR. This seems to confirm that PR has fixed intervals after which its SUPERPROXY contacts a new GATEWAY or ends the communication. On the other hand, OL and SM try to connect to the server with a sequence of machines and close the connection only when the last GATEWAY terminates its attempts.

The third column of Table 5.2 gives the number of different IPs that produced SYN packets per communication. We can notice that all the providers use on average 3-4 different IPs to contact the server. This shows an inner protocol of the RESIP providers. Apparently, when a GATEWAY is unable to

Table 5.4: Median intervals between requests of different IPs in the same communication

RESIP	1st-2nd IPs	2nd-3rd IPs	3rd-4th IPs	4th-5th IPs
OL	7s	7s	6s	5s
PR	3s	6s	5s	5s
SM	0s	6s	7s	6s

establish a TCP connection, they believe the problem is with the GATEWAY. Thus, they retry from another machine. Table 5.3 shows the mean value and standard deviation for the average number of distinct countries and continents of the IPs selected for each communication. We can notice that when the RESIP chooses a new machine, the IP belongs generally to a country different from one of the previously used machines for that communication. In most cases, the continent is also different. This is probably a sign that RESIP providers check if there are regional connectivity problems and/or the server is blocking requests from specific areas of the world.

Table 5.4 shows the median of the INT_DIFF_IPs, the interval of time between an IP contacting our server and the appearance of a new IP in the same communication. We can see that intervals of OL have similar values. For PR, the first interval is roughly half of the subsequent ones, which present similar values. SM shows similar values for the intervals except for the first one, in which the difference is set to zero. In the dataset, the granularity of the intervals is in seconds, so this means that, either the two SYN packets were issued from the provider at the same time, or the interval between them is less than one second. We believe the second to be a more plausible explanation since sending two SYN packets for each connection would not be efficient for the providers.

These values of the interval between requests from different IPs in a single communication confirm the idea of an automation process at the RESIP provider. In one case, requests from each new IP are sent after a similar amount of time. In the other two, the interval between the first and second IPs is shorter. Subsequent intervals present constant values.

In the fourth column of Table 5.2, we find the average number of original SYN packets sent per IP in a communication. This corresponds to the average number of ports used by a device to send SYN packets for that communication. The fifth column of the same table gives the average number of retransmitted

SYN packets for each of the IPs that contacted the server during the duration of the communication. As explained in Section 3.5, when a server does not send an ACK message, a generic client keeps retransmitting the SYN packet using the exponential backoff behavior. The GATEWAYs of PR appear to behave in this way, since the average number of SYN packet is close to one and the same packet is retransmitted more than once.

This is not the case for both OL and SM. For the communication performed with these two providers, a single IP sends SYN packets to our server much more often by opening a new connection from a new port than retransmitting the same packet. Moreover, studying the distribution of the BEHAVIORS_COUNTS attribute, we see that there are no cases where an IP only retransmits packets and does not start also a new connection.

The MEDIAN_NEW_SYN parameter tells the median delay between new SYN packets from different ports of the same IP. The median of these values is 1.5s for OL and 1s for SM. In the kernel exponential backoff behavior, the minimum interval before a retransmission is usually 1s. Thus, it is more probable for OL SYN packets, that have a median interval greater than this minimum interval, to be retransmitted, as shown by our data (column 5 in Table 5.2).

5.4 Detection of Second MRP Discussion

In Section 3.5, we introduced the MRP_DETECTION technique. When a RESIP provider presents the second MRP and tries to contact a non-acknowledging server, its GATEWAYs send multiple SYN packets from different ports to the server. Thus, an acknowledging server can simply delay the sending of each SYN-ACK packet and check if the client sends a second SYN packet from another port. In that case, the connection is likely proxied through a RESIP[21].

We can apply this technique only for RESIP connections presenting the second MRP. The analyses performed in Section 5.3 show that OL and SM GATEWAYs consistently present the second MRP. For this reason, we can say that the MRP_DETECTION can identify them.

When applying this technique on connections, the server needs to delay the sending of the SYN-ACK packets. As explained in Section 3.5, this results in delays on the client side, since the server needs to wait enough time for the GATEWAY to send a second SYN packet. The analyses show that the median delay between the first and the second SYN packets coming from the same IP but different ports is 1.5s and 1s for respectively OL and SM. Hence, the server should wait for at least 1.5s to detect half of those. According to [125],

[21] Unless a corner case, such as the ones described in Section 3.5, takes place.

any delay higher than 1s degrades the user experience. For this reason, the MRP_DETECTION technique likely affects real users' experiences.

Therefore, our suggestion is to use it in combination with the RTT_DETECTION, presented in Section 3.4, to achieve attribution and mitigation. In this combined detection, we can redirect the connections declared as RESIP with the RTT_DETECTION approach to another machine. We can perform the redirection with the HTTP REDIRECT (HTTP 307 Temporary Redirect) to another machine. This obliges the client to open a new connection with that machine. In case of a RESIP connection, the GATEWAY (or a new GATEWAY depending on the settings) starts a new TCP session with the new machine where we can perform the MRP_DETECTION.

In this way, only a percentage of real users (the false positives of the RTT_DETECTION i.e. 17.66±7.53% of the real-world traffic as discussed in Section 4.4) would be impacted by the delay. Moreover, we delay all RESIP connections providing a mitigation solution, similar to what the Imperva Tarpit does [33]. Finally, we are able to distinguish the connections of different RESIP providers.

5.5 Summary

In this chapter, we present a measurement campaign that characterizes RESIP providers contacting a non-acknowledging server.

Our results show that all the analyzed providers use 3-4 GATEWAYs to try to reach the server. Moreover, these machines are usually located in different countries and continents. This shows that RESIP providers try to circumvent possible regional problems and restrictions using GATEWAYs in other areas of the world.

Furthermore, two of the analyzed providers consistently show the second MRP and are thus good candidates for the MRP_DETECTION technique.

Based on our measurements, the delay introduced by the technique to identify the RESIP providers would generally affect the real users' experiences. For this reason, we propose to use it in combination with the RTT_DETECTION technique (Section 3.4).

Thanks to the RTT_DETECTION and MRP_DETECTION techniques, we are able to detect RESIP connections and differentiate them from genuine ones. Moreover, while RESIP providers could evade these detection techniques, this would require them to perform significant changes in their infrastructure. Hence, these two techniques enable us to answer the first research question that this thesis aims to answer (RQ1, Section 1.1). In the next part of this thesis, we will address the other proposed research question thanks to our deceptive approach for scraping mitigation.

Part II

Web Scraping Mitigation Through Deception

Chapter 6

Web Scraping Mitigation through Redirection to a Honeypot

6.1 Introduction

As explained in Section 2.2, web scraping is a problem for e-commerce websites. These websites receive large amounts of scraping requests which cause losses of revenue and additional costs for the providers. Moreover, the current detection and mitigation of scraping requests are not effective. The detection is performed on parameters whose values scrapers can easily change and are similar to the ones of legitimate users. Regarding mitigation, scrapers can infer they have been detected from the side effects of the current techniques we use for this task. In this scenario, scrapers simply identify the parameters we use to detect them and they change them to avoid the mitigation. Then, they continue their activity until they are detected again. At that point, they change again the identified parameters and the process repeats itself.

In this chapter, we present a new mitigation approach that overcomes these limitations and enables us to break this infinite loop. It consists of redirecting the scraping requests to a web application honeypot. There, scrapers receive pages with the same structure as the original website but with modified content. In this way, scrapers cannot understand that their requests are mitigated and, at the same time, they do not access the real content of the website. Moreover, the honeypot enables us to poison the information they scrape and on which they monetize, producing financial damage on their side. Furthermore, with this platform we can study the semantics of the payloads and the distribution of the IP addresses of specific bots, giving us new insights into their ecosystem.

In late 2019, in the context of the Master's thesis project of the author, we performed a first case study with the honeypot platform in collaboration with Amadeus IT Group (Amadeus). In Section 2.2.3, we shared background information about web scraping against this company and we showed the challenges of its analysts in detecting and mitigating scraping traffic.

In the first case study, scrapers leveraged a corner case in the syntax of their requests to detect when we were redirecting them to the honeypot. They sent queries with an erroneous combination of secondary search constraints. The real website answers this type of query with an error page. Instead, our honeypot was sending back the result of the search. The scrapers used this information to understand which parameters we were performing detection on and change them to avoid identification. Thus, our case study ended prematurely. Even if the campaign stopped in an early stage, the honeypot platform enabled us to acquire precious insights about the scraping bot ecosystem (more details are available in [126]). For this reason, we decided to perform another attempt, after modifying the implementation to cover the corner case previously discovered.

In the next sections, we provide a short recap of the setup of the Honeypot platform (Section 6.2). We refer to [126] for a more complete explanation. Then, we propose the analysis of the results of the second case study we performed with the honeypot. Section 6.3.1 studies the semantics of the requests performed by scrapers. We reveal the automated process behind those requests and we show that the majority of the requests belongs to one single bot campaign. In Section 6.3.2 we characterize the IP addresses from which the requests were generated. We show the distribution, reputation and repetition patterns. We explain how these addresses most likely belong to a RESIP service. Finally, Section 6.3.3 discusses the value of the honeypot as a mitigation technique and its success in luring the attackers.

The different analyses of the honeypot data proposed in this chapter are available in the following publications: [C6], [C7], [J1].

6.2 Setup and Methodology

Amadeus provides software solutions to help airlines handle their booking processes (Section 2.2.3). For our case study, we consider a specific product among the ones offered by Amadeus, which we refer to as *Product* X. When an airline company chooses *Product* X for its booking domain, it builds a separate front-end website.

Figure 6.1 shows the flow of the requests when using this product. The

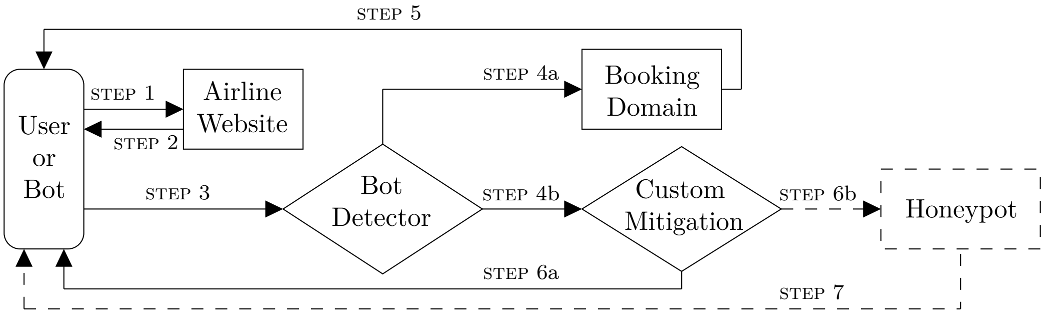


Figure 6.1: Flow of actions performed when a User/Bot searches flights using *Product* X.

STEPs in the figure are explained hereafter. Initially, the initiator of a connection (User/Bot) reaches the front-end website of the airline (STEP 1) and receives a webpage in return (STEP 2). This webpage contains a form where the users insert the constraints of their searches e.g. departure and destination locations. At this point, the initiator sends the form to the Bot Detector (STEP 3). If the Bot Detector concludes that the parameters of a request do not match any bot signature, it redirects it to the Booking Domain (STEP 4a). This domain finds all the available flight solutions matching the search and sends them to the initiator (STEP 5). Hence, in this case, the initiator receives a successful answer. On the other hand, if the Bot Detector finds a matching bot signature (STEP 4b), it performs the Custom Mitigation associated with that signature. For example, the bot can receive a CAPTCHA [28], a JavaScript challenge or the request can be blocked (STEP 6a).

In our setup, we have implemented a new type of Custom Mitigation. It consists of a transparent redirection of the request to a honeypot (STEP 6b). This platform, external to the IT provider environment, can provide answers to requests while modifying the prices at will. The template of the real page is obtained periodically from the real booking domain. Fares are retrieved thanks to a *Fares API* of Amadeus. The system can modify the fares, insert them into the template and send the response back. In this way, bots receive a response with the same syntax but prices different from the original ones (STEP 7).

To experiment with our honeypot, we collaborated with a specific airline company that uses *Product* X. We call it *Airline* X. At the time of the case study (beginning of 2020), *Airline* X was receiving an average of 1 million daily booking requests. The Bot Detector mitigated close to 40% of them. We configured the Detector to forward to our honeypot all the requests matching a specific bot signature.

We used the following criteria to choose the signature:

- The signature was not receiving any custom mitigation yet, to prevent scrapers from seeing drastic changes in the type of received responses
- The signature presented characteristic daily peaks of activity, to be able to effectively see any change in the behavior of the bot
- The traffic associated with the signature could be entirely diverted to the honeypot, considering the limitation we had in using the *Fares API*[22]

In order to respect these constraints we chose a signature that presented only daily peaks of activity (300-500 requests) in a 40 minutes time window at the same moment of the day.

The case study started on January 7th, 2020. We served the bots with unmodified fares in the first 72 hours. The objective of this phase was to test that our modified setup was not creating particular artifacts that could be detected by the bots. Then, we started the luring phase. We randomly modified the fares of 10% of the requests by 5%.

The requests matching the bot signature drastically disappeared on March 3rd, 2020. We believe the reason is linked with the business needs of the actor behind these bots. Indeed, that date coincides with the beginning of the worldwide COVID-19 pandemic. Furthermore, *Airline X* is the main airline for a country whose government issued its first major travel restriction on the 2nd of March, practically ending airline travel to and from that country. Without any flight to search for to/from that country, there was no reason for the malicious actor to keep scraping the website. This and the fact that the bot behavior did not change after the increase in the prices give us confidence that the disappearance of this bot campaign was caused only but external factors.

6.3 Honeypot Results

6.3.1 HTTP Payload Analysis

In this section, we propose an analysis of the HTTP payloads of the 22,991 requests received during the 56 days of the case study. Since the Bot Detector does not consider the payloads to build bot signatures, the goal of our examination is to understand if the bot traffic redirected to the honeypot

[22] Amadeus imposed a maximum number of requests per minute we could perform with the *Fares API*.

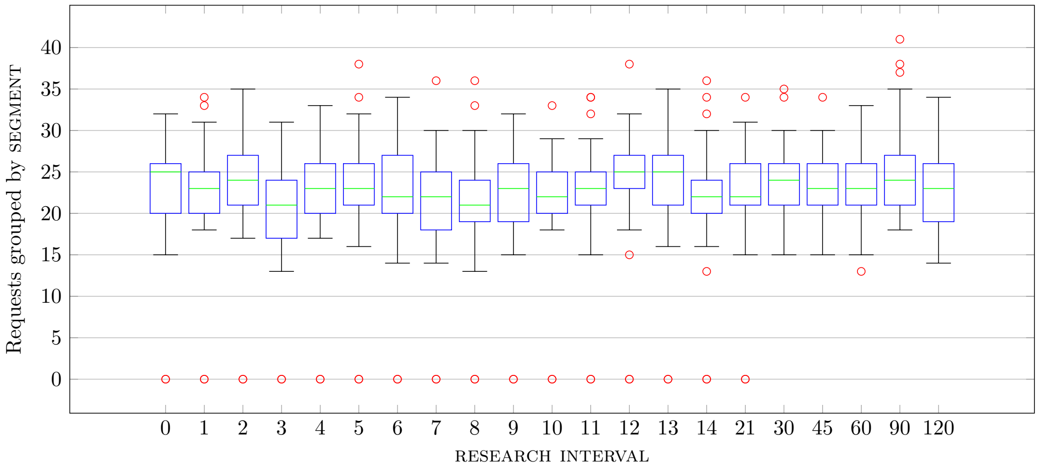


Figure 6.2: Count of requests grouped by SEGMENT over each RESEARCH INTERVAL.

corresponds to one or more scraping campaigns. Indeed, multiple scrapers could be using the same parameters on which the fingerprint is built and it is important to define the different contributions of each party. Moreover, this analysis enables us to gather more insights into how scrapers perform their scraping campaigns. This information can help us in assessing the efficacy of using the honeypot to lure attackers.

Inspecting the payloads, we see that the requests ask for a return flight (resp. one way) in 51.5% of the cases (resp. 48.5%). In the case of a return flight, the return date is always 7 days after the departure. The 22,991 requests only look for 25 combinations of 16 departure and 12 arrival airports from where this airline operates flights, an extremely small fraction of the airline offers. This data is consistent with the idea of a repetitive data collection task and that there is an automated process behind these requests. Moreover, the time analyses presented hereafter support these conclusions.

For each request, we define the time interval between the date on which the request is performed and the date of the departure of the searched flight as RESEARCH INTERVAL. Studying the RESEARCH INTERVALs among all requests we see that they present a specific pattern. The vast majority (99.8%) take values either between 0 and 14 days or 21, 30, 45, 60, 90, 120 days. We define these values STANDARD RESEARCH VALUEs.

We call SEGMENT the combination of the type of flight (one-way/return), departure location, and arrival location included in it. Multiple requests can present the same SEGMENT. In Figure 6.2, we represent the count of requests, grouped according to the corresponding SEGMENT, for each

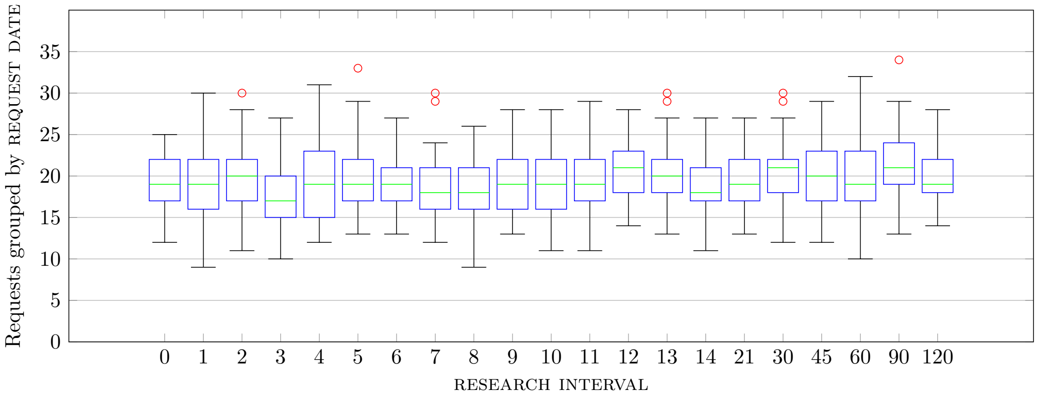


Figure 6.3: Count of requests grouped by REQUEST DATE over each RESEARCH INTERVAL.

RESEARCH INTERVAL in the STANDARD RESEARCH VALUEs. On the x-axis, we see the value of the RESEARCH INTERVAL, and on the y-axis the number of requests. To build each boxplot, we first calculate the number of observed requests for each SEGMENT and RESEARCH INTERVAL. Then, we calculate the median and percentile statistics considering all the SEGMENTs and a specific RESEARCH INTERVAL. This figure clearly shows that, for each RESEARCH INTERVAL, the median of requests presents stable values for the majority of the requests. Thus, the bots request equally all the considered RESEARCH INTERVALs. We see that for the RESEARCH INTERVALS lower than 30, we have outlier SEGMENTs which present count zero. Indeed, there are two segments that are queried just for RESEARCH INTERVALS equal to or higher than 30. Apart from this, each boxplot shows small variability. This tells us that each SEGMENT is queried with similar contributions for each RESEARCH INTERVAL.

Figure 6.3 looks at this regularity from a different angle. For each request, we define REQUEST DATE as the date on which each request is performed. Similarly to the previous figure, the x-axis represents the RESEARCH INTERVALs, the y-axis the count of requests. We build each boxplot by grouping the requests according to their RESEARCH INTERVAL in the STANDARD RESEARCH VALUEs and counting the occurrence for each REQUEST DATE. The new boxplots show a high regularity in the querying process. Every day similar kinds and amounts of queries are performed.

The regularity of the results of these time analyses shows us, once again, strong automation behind the scraping requests. Every day, bots request similar combinations for similar time frames. For this reason, we can conclude that most requests correspond to one single scraping campaign that shows specific characteristics. Only a few requests (48, 0.2% of the total) present values of the RESEARCH INTERVAL different from those in the STANDARD RESEARCH VALUES (20, 31, 44 days). We note that these requests were performed outside the daily 40 minutes window. We can conclude that they most likely correspond to a different small campaign. Another possible explanation is that the scraper behind the main campaign is performing small checks for specific information outside the normal scraping window.

Studying the content of each request, we can see that the queries performed day after day by the scraper are not identical. If we compute the 4 tuples made of i) departure airport, ii) arrival airport, iii) time interval, iv) type of flight (one way or return), we can identify 982 distinct ones over the whole case study. If we compare this with the average number of requests per day (410), we can conclude that not all tuples are queried every day. On average the scraper queries each of them 23.41 times during the whole running time of the case study. Generally, he asks for a specific tuple just once a day, but 20% of the tuples, at least on one occasion, are registered more than once a day. The maximum number of times the same tuple has been requested on the same day is 8. Almost every day at least one tuple has been queried more than once. Requests asking for the same tuple are always performed in a short period of time. The maximum interval between two of them is 337 seconds (around 5 minutes and a half), the minimum one is 5 seconds. The average value is 45.2 seconds.

This behavior is somehow peculiar. Not only the scraper asks for the same combinations multiple times, but it also does it in a short amount of time. One possible reason is that he is performing a check of consistency for the prices. Another possible explanation could be that a bot did not return an expected reply quickly enough, causing its master to ask another bot to submit the same request a second time. In Chapter 3, we saw that Residential IP Proxies used by scrapers implement this last strategy. This tells us that parties of the sector implement this approach and thus seems to confirm that the same combinations are requested multiple times because of delays in the answer of a bot.

6.3.2 IP Addresses Characterization

The 22,991 requests collected in the honeypot have been sent by 13,897 unique IPs. We use the term "honeypot IP addresses" to identify them in the rest of this work. In this section, we study their distribution, reputation and repetitions. We show how the results of our analyses suggest that these IPs are provided by Residential IP Proxies (RESIP) services.

Table 6.1: Honeypot IPs matches with IPs in external datasets.

(a) Dates in which a honeypot IP performed a booking, corresponding appearance dates in the honeypot and time passed between the two dates.

Bookings	Honeypot	Δt
2020-01-17	2020-02-01	15d
	2020-02-05	19d
	2020-02-14	28d
2020-02-26	2020-01-10	47d
	2020-01-23	34d
2020-02-29	2020-02-01	28d
2020-02-06	2020-02-23	17d
2020-02-07	2020-01-24	14d
	2020-02-02	5d
	2020-02-19	12d

(b) Matches between Tor exit nodes and honeypot IPs.

Date	# IPs	% IPs
2020-02-14	12	0.09%
2020-02-15	24	0.17%
2020-02-18	20	0.14%
2020-02-19	40	0.29%
2020-02-24	12	0.09%
Total	108	0.78%

Distribution

The honeypot IP addresses belong to 1,187 /16 blocks. This means that, on average, there are less than 12 IPs used within each /16 block (less than 0.02% of that IP space). Furthermore, the geo-localization of these addresses indicates 790 distinct origins, in 86 different countries. This highlights how widespread the distribution of these IPs on the Internet is while they participate in a well-coordinated scraping campaign, as described in the previous section.

Reputation

To define the reputation of each honeypot IP address we check if the IP performed benign and/or malicious activities outside of the honeypot environment during the time of the case study. To do so, we take advantage of external datasets.

First, we focus on benign activities. We analyze if, during the running time of the case study, any of the honeypot IPs was associated with a completed booking on *Airline* X or other airline domains managed by Amadeus. Five such IPs reserved a flight in that time frame. Table 6.1a shows the dates on which the bookings were performed and the dates on which we see the corresponding IP in the honeypot logs. The dates differ among them and

Table 6.2: Analysis of the honeypot IPs with *IPQualityScore*.

(a) Classification of the IPs.

Type	# IPs	% IPs
VPN	9,138	65.76%
Proxy[23]	1,075	7.74%
Recent abuse	3,878	27.91%
Bot Status	2,780	20.00%

(b) Distribution of the fraud score.

Score (S)	# IPs	% IPs
$S < 75$	3,958	28.48%
$S \in [75, 85[$	6,371	45.84%
$S \geqslant 85$	3,568	25.67%

none of the booking requests presented the bot signature associated with redirection to the honeypot. This tells us that, during the scraping campaign, at least part of the honeypot IPs have been used by legit users. Moreover, most likely these users were different actors from the one performing the scraping campaign.

We know that an IP address can be associated with more than one device, e.g. when multiple devices are behind a Network Address Translation (NAT), and that the same IP address can be associated with different devices at different points in time. For this reason, we cannot conclude that legit users and scrapers were taking advantage of the same devices. However, since these IPs performed legitimate activities at some point in time, we can say with confidence that they are not datacenter IPs. Thus, we have strong indications that at least a part of the honeypot IPs does not belong to datacenters.

Using the Python library Pydnsbl [127], we can see that 76% of the honeypot IPs have been blocked in at least one blocklist at the time of our analysis (July 2020). Regarding Tor, 72 honeypot IPs were announced as exit nodes on the same day of their registration in the honeypot. This happened on 5 different days. Table 6.1b shows the matches between the Tor exit nodes and the honeypot IPs. It is peculiar how only 5 days close in time have witnessed these matches. We hypothesize that the scraper tried to take advantage of the Tor nodes but did not continue with this strategy.

Table 6.2a shows the results of the analysis of the honeypot IPs with *IPQualityScore*. This service tells if an IP has been used in "automatic fraudulent behavior" thanks to the "Bot status" category. Moreover, it indicates a positive value of "Recent Abuse" if the IP has been involved in a "recently verified abuse across their network". The "abuse" behavior

[23] From the total number of positive matches, 10,213, we subtract the number of positive values for the category VPN.

includes performing a chargeback, being detected as a compromised device, and being associated with fake app installation. The "VPN" category indicates IPs that allow tunneling and the "Proxy"[24] one identifies devices infected by malware, users selling bandwidth from their connection, or other types of proxy like SOCKs, Elite, Anonymous, Tor, etc. We can see that a large fraction of the collected IPs is associated with proxying out requests. Furthermore, *IPQualityScore* provides a general fraud score for each IP: this value ranges from 0 to 100, indicating a suspicious activity when higher than 75 and a high risk when greater than 85. Table 6.2b tells us that around 72% of the honeypot IPs show suspicious behavior and that 26% are classified as high risk.

These analyses tell us that the IPs collected in the honeypot have been associated with malicious activities. Moreover, they tell us that a large percentage of them has been associated with proxying activities.

Repetitions

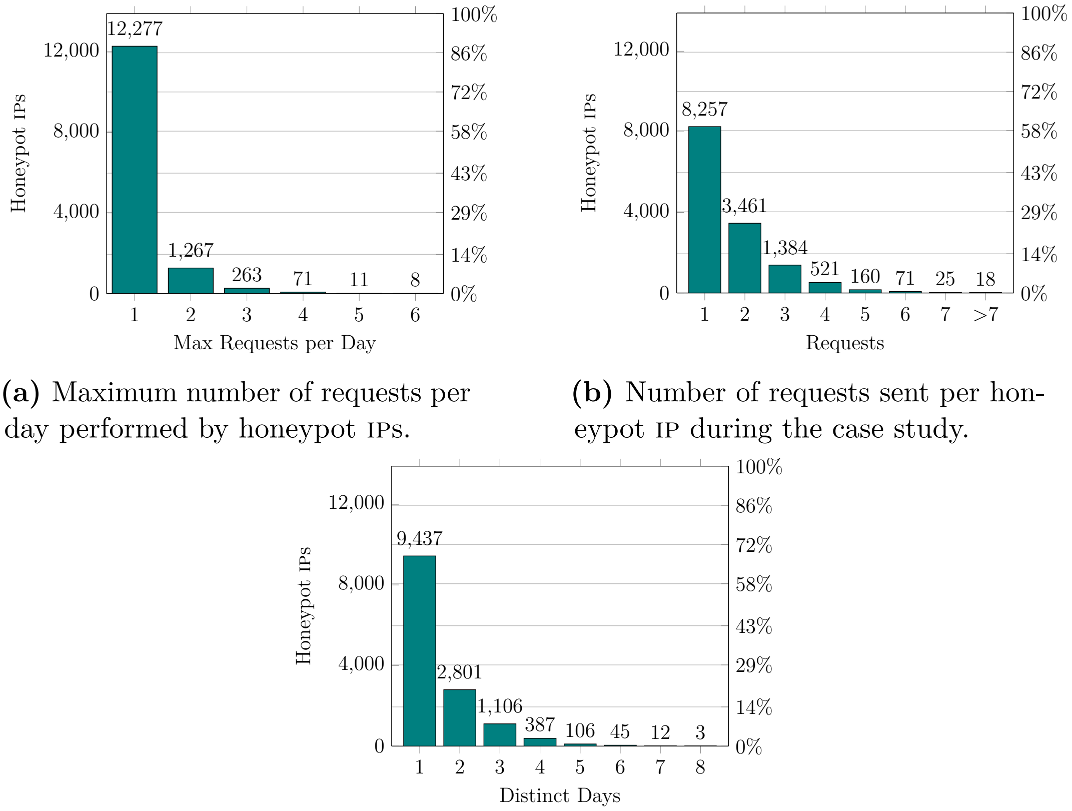
Figure 6.4a shows the maximum number of requests per day sent by each IP address. Almost all the IPs (98.06% of the total) make at most two requests per day. We can see that 12,277 IPs (88.94% of the total) make only one request per day. This data tells us that the bots of this campaign can be categorized as Advanced Persistent Bots. These bots normally perform just one request per IP address per day [9].

However, looking at the whole running time of the experiment, we see that the scraper reuses some IPs. Figure 6.4b shows the total amount of requests made per distinct IP over the whole case study. Here, we see that 8,257 IPs have sent only one request. If we compare this value with the 12,277 of Figure 6.4a, we understand that a large number of IPs shows up on at least two different days, issuing a single request every time. This is confirmed by Figure 6.4c where we see the number of distinct days in which an IP performs a request to the honeypot. Almost 30% of the IPs contact the platform on at least two different days. This information suggests that the scraper behind the campaign tried to have a new IP for each request but had a limited pool of IPs.

Discussion

These analyses show that the IPs contacting the honeypot are widely distributed around the world. They are largely used for proxying activities and we have strong indications that (at least) a part of them does not belong to

[24] A "VPN" is automatically a "Proxy" according to their definitions.



(a) Maximum number of requests per day performed by honeypot IPs.

(b) Number of requests sent per honeypot IP during the case study.

(c) Number of distinct days on which the same IP contacted the honeypot.

Figure 6.4: Analysis of the repetitions of the honeypot IPs.

datacenters networks. Moreover, they are associated with malicious activities in various contexts. For these reasons, we strongly believe these IPs are part of a RESIP network. Indeed, RESIP providers proxy requests thanks to residential devices located all over the world. At the same time, they are also associated with malicious activities [38], [45], [46]. The collected IPs show matching characteristics with this description.

RESIP providers claim to have access to a pool of tens of millions of IPs that can be used as exit points of requests. If the honeypot IPs were part of one of these networks, the repetitions exposed in this section would suggest that the pool at the RESIP disposal is much smaller than the advertised values. Indeed, 2,801 IPs have been used twice during the case study and this is inconsistent with picking randomly from a large pool of millions of IPs. Calculating the probability that a given IP got picked twice over this period comes down to resolving the classical birthday paradox. We can generalize it as follows:

> *Given n random integers drawn from a discrete uniform distribution with range [1,B], what is the probability p(n, B) that at least two numbers are the same?* [128]

In our case, n is equal to 56, the number of days where IPs from the pool were assigned to the scraping campaign. The variable B, set to 365 for the birthday paradox, is the size of the RESIP pool from which the picking is performed.

The formula $1 - (\frac{B-1}{B})^{\frac{56(55-1)}{2}}$ gives an approximate result for the birthday paradox problem. Using it with our data, we obtain the probability that an IP is drawn twice over the period of 56 days. Computing the probability and varying the size of B, we obtain:

- If $B = 10,000,000$ then $p(56, 10,000,000) \approx 0.000154$
- If $B = 1,000,000$ then $p(56, 1,000,000) \approx 0.001538$
- If $B = 100,000$ then $p(56, 100,000) \approx 0.015282$

From these results, we can see that the probability of having a repetition with large sizes of B is extremely low. Considering that we have seen more than 30% of the IPs drawn at least twice, either B in our case study is significantly lower than what RESIP services announce or the assignment of IPs is not randomly done, or both. Knowing this information is essential to perform the detection on the server side. If only a small pool of IPs or a small portion of a large pool is used to scrape a website, we could take advantage of IP blocklists to perform detection.

In Chapter 7, we further explore this line of research. We build a mathematical model to find the pool size of RESIP providers starting from the IPs used as exit points. We apply this model on the honeypot IPs and IPs collected from 3 RESIP providers. While in both cases the resulting pools are smaller than the advertised ones, we reveal that the one associated with the honeypot IPs is much smaller than the ones of the studied providers. This tells us that if the honeypot IPs were indeed part of a RESIP network, the provider was different from the other ones we analyzed and had a much smaller pool of IPs at its disposal.

6.3.3 Luring Attackers as a Mitigation

The case study ran between the 7th of January 2020 and the 2nd of March 2020, for a total of 56 days. The platform served modified fares to the bots for 53 days. Over the duration of the case study, the honeypot received 22,991 requests. Thanks to the analysis of Section 6.3.1, we can say with high confidence that the requests were part of a single scraping campaign.

The daily average amount of sent queries was 410±33 queries. The vast majority of the collected requests arrived in the same daily time window with a duration of 38.18 minutes on average. The amount and the timing of the requests are in line with those of that bot signature before the beginning of the case study, even after the beginning of the modifications. Thus, we can conclude that the scrapers did not realize they were receiving modified content and did not change their behavior accordingly.

In our case study, we have successfully changed randomly some of the values without causing the bots departure. From this, we have learned that the scraper behind this campaign i) has no ground truth to compare the returned values and/or ii) his plausibility checks (as the ones potentially seen in Section 6.3.1) are not sophisticated enough to detect small changes, not even by correlating values collected over several days.

In our case study, the requests were produced by a single bot signature corresponding to a single campaign. Moreover, they were collected only for a finite period of time. Hence, we know that our analysis can not pretend to be directly representative of the whole bot traffic. However, they provide a good picture of the behavior of this particular botnet and, from our results, it could be easily generalized.

We redirected to the honeypot requests which were produced by scrapers aware to be targeting a protected system. Amadeus has suffered from bot attacks for many years and has used different mitigations to stop them. Thus, bots choosing to send requests there know they most likely encounter some type of mitigation. This tells us that the studied botnets were sophisticated enough to try to overcome leading commercial anti-bot solutions.

Moreover, as seen in Section 6.3.2, the bots producing the honeypot requests can be categorized as Advanced Persistent Bots, the highest sophistication category of bots used to perform scraping campaigns. Furthermore, we have strong hints that the IPs producing the requests are part of a RESIP network. These services are used more and more to perform scraping. This information tells us that the campaign we studied is a representative example of the scraping campaigns category. Hence, this gives us hope that our results could be easily generalized.

The results presented in this section helped us convince Amadeus to move forward into developing a honeypot environment built in their production system and the work is underway. In this solution, cache prices could be served to the bots at a cheap price for the provider. Cached values dramatically reduce the cost of computing the responses. Some sensitivity analysis remains to be done to assess the cache refreshing rate.

6.4 Summary

In this chapter, we have investigated the possibility of mitigating Advanced Persistent Bots thanks to the redirection to a honeypot. Our honeypot platform is able to produce responses that are, syntactically, indistinguishable from the real ones, but that contain modified prices. In this way, scrapers do not realize they are not reaching their goal and we can poison the information they collect.

Even if the mitigation was applied to a single bot campaign, our analyses of the payloads and IP addresses suggest that this botnet is representative of the ones performing web scraping. Thus, our results could be generalized. Thanks to this, we have convinced our partner, Amadeus, to move forward into developing a honeypot environment built in their production system and the work is underway.

The results of this section give us the means to address the second research question proposed by this thesis (RQ2, Section 1.1). Moreover, the honeypot enabled us to see first-hand the extent to which scrapers use Residential IP Proxies. As explained in Section 2.3, scrapers take more and more advantage of RESIPs to perform their activities. This motivates us to conduct an extensive study of these parties and their ecosystem, as proposed in the next chapters.

Part III

Residential IP Proxies and Scrapers Ecosystem Analysis

Chapter 7

Analysis of the Residential IP Proxies Ecosystem

7.1 Introduction

Section 2.3 explained how different actors use RESIPs for malicious purposes and Section 2.2.3 show their involvement in scraping activities. It is important to characterize the RESIP ecosystem to better understand it and find new ways to mitigate the threats it represents when used by scrapers and other malevolent parties. In this chapter, we take advantage of the 110 days dataset of RESIP connections we presented in Section 4.2.4 (RTT_DS) and we use it to disclose new findings about RESIP inner functioning and try to assess the order of magnitude of their pool of residential IPs. Moreover, we take advantage of the analyses presented in this chapter, Chapter 3, Chapter 4 and Chapter 5 to have a complete overview on RESIP and draw conclusions on what is known so far about these parties.

To collect RTT_DS, each client performs the DNS resolution for the SUPERPROXY domain name and sends a HTTP CONNECT to the obtained IP address. The SUPERPROXY forwards the request to a GATEWAY that finally sends it to the server. All the requests sent by our clients share the same customer ID. For this reason, it could be possible that the SUPERPROXY and GATEWAY IPs we collect are only taken from a partition of IPs assigned to our account. However, based on the available documentation and the interactions we had with the companies, there is no indication that this is a practice they follow. Because of this, we can safely assume that our dataset is a representative sample of the global pool of IPs belonging to these parties. Thus, our results describe the characteristics of the whole pool available to the providers.

Thanks to an in-depth analysis of the SUPERPROXY and GATEWAY IPs in

Table 7.1: IPs distribution statistics per provider

RESIP	# Connections	# Countries	# /32	# /24	# /16	# /8	# AS
BD	2,413,405	226	1,546,886	712,274	23,274	193	17,026
OL	22,387,788	226	6,660,452	846,165	15,230	194	19,370
PR	22,523,876	234	3,982,149	411,949	14,145	201	9,871
SM	22,353,578	224	6,852,898	859,946	15,288	194	19,501

RTT_DS, we disclose novel insights about the inner working of RESIPs in terms of geographic distribution, types and management of IPs used. Moreover, we show the similarities and differences among the four studied RESIP services.

Furthermore, we use the RTT_DS to discuss the size of the pool of residential IP addresses that RESIP providers have at their disposal. While these providers claim to have access to a large amount of IPs, the analyses in Section 6.3.2 seemed to suggest otherwise. In this chapter, we reach the same conclusions by estimating the pool size through mathematical modeling.

In the following sections, we first present the novel findings about RESIP providers (Section 7.2). Then, we propose our analyses concerning the IP address size of the RESIP (Section 7.3). We conclude the chapter with a discussion on the lessons learned from studying RESIP providers (Section 7.4).

The works presented in Section 7.2 were published in [C5]. The model illustrated in Section 7.3 was presented in [C7], [J1], while its application on the RTT_DS was included in [C5].

7.2 Findings about RESIP Inner Functioning and Relationships Among Providers

In this section, we study the IP addresses collected in RTT_DS to reveal new findings about their inner functioning and the relationships among different providers. During the 110 days of collection, we registered 22M+ connections for Oxylabs (OL), Proxyrack (PR), Smartproxy (SM), as shown in Table 7.1. Bright Data (BD) count is lower than the other entries due to the discontinuation of the service imposed by the company (see Section 4.2.1). Moreover, Table 7.1 shows the wide distribution of the IPs associated with the requests in terms of countries, Autonomous System (AS) and subnets. We notice that BD shows high variability even if we have fewer connections from this provider. This tells us that potentially its real distribution presents even

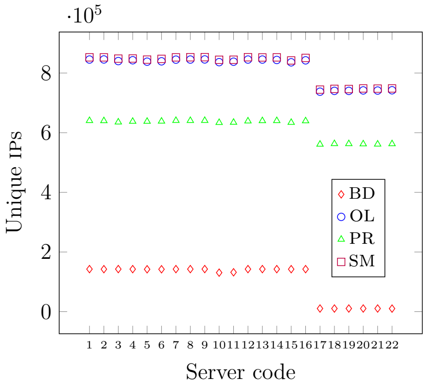


Figure 7.1: Unique GATEWAY IPs registered by each server.

Table 7.2: IPs repetitions per provider

RESIP	Repeated IPs	Repeated IPs per server	Repeated IPs per client
BD	31%	3±1.6%	3.3±1.8%
OL	49%	16.3%±0.5%	16.3%±1.3%
PR	61%	23%	23.4%±0.2%
SM	49%	15.7±0.4%	15.7%±0.4%

higher values. For PR, instead, we see the lowest values in terms of /32, /24 and /16 subnets count and AS variability. At the same time, it presents IPs in more countries than the other providers and it shows the highest value in /8 subnets diversity. This shows that this provider has a better presence on the global scene but lower shares in each region.

Thanks to the analysis of the above-mentioned connections, we introduce some novel findings into the inner workings of RESIP providers and their relationships. We present these findings, numbered between F1 and F7, in the following sections.

7.2.1 F1: Assignation of GATEWAY IPs to Minimize Repetitions per Path

Fig. 7.1 shows the amount of unique GATEWAY IP addresses registered by each server for every provider. On the x-axis, we see the codes (1-22) of our servers. On the y-axis, we find the count of unique IPs. Each mark represents the count of unique GATEWAY IPs seen by a server for a specific provider. Codes 17-22 correspond to the machines added in the second phase of the experiment. They have received fewer requests than the others and this is reflected in the count of unique IPs.

In general, we can see that, for each provider, the amount of unique IPs contacting each server is constant. This shows an inner strategy of RESIP providers. These parties try to assign unique GATEWAY IPs to each server with the same proportion.

Moreover, we can notice that if we sum the amount of unique IP addresses registered at each server for a specific provider we obtain much higher values than the total amount of unique IPs (# /32 in Table 7.1) seen for that provider across all servers. For instance, looking at PR, the average number of unique IPs in Fig. 7.1 is around 600,000. Multiplying this value for the number of servers (22) we obtain 13,200,000 which is much higher than the number of unique IPs seen for the same provider in the whole experiment (3,982,149).

Examining the count of unique IPs per path (combination of client-server), we see that the number of unique IPs per path is 4971±4252, 45526±4247, 43164±3929, 45552±4252 for, respectively, BD, OL, PR and SM. If we multiply these values by the total number of paths (22*22), we obtain again values much bigger than the ones in the fourth column of Table 7.1 (e.g. for OL, 22,034,584 instead of 6,660,452).

These results tell us that RESIP providers increase the variability of IP addresses for a single path, and try to optimize their stealthiness by giving each client a new IP for each of its requests, even if it queries a distinct server. At the same time, they reuse much more often the same IPs for other client-server combinations.

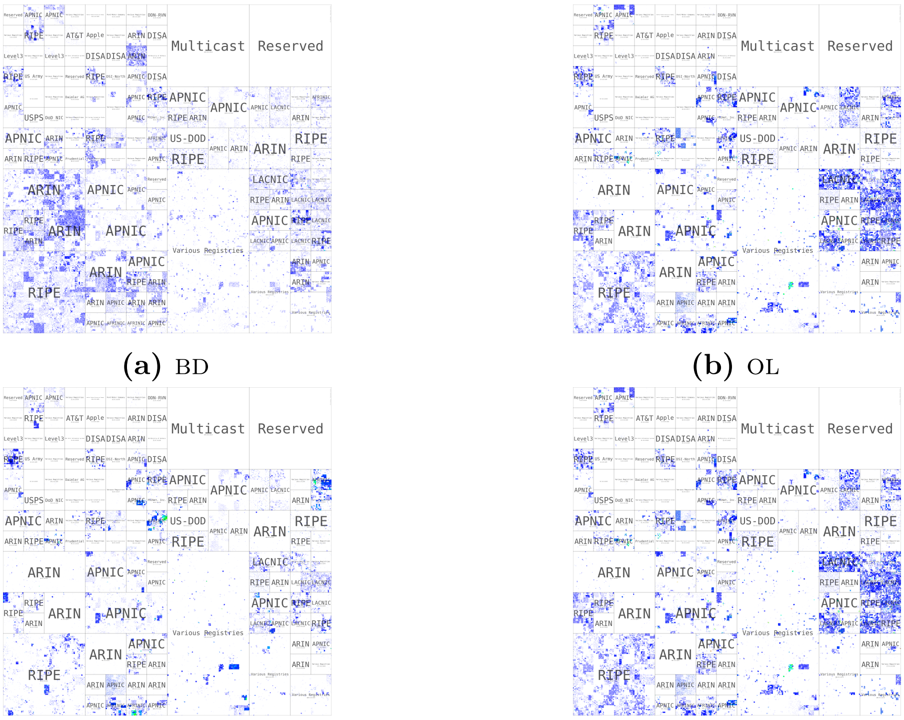
Furthermore, the statistics about the repetitions of IPs, as shown in Table 7.2, validate this idea. We define a repetition when an IP address shows up in at least two connections. The third last column shows the percentage of repetitions with respect to the number of unique GATEWAY IP addresses.

PR presents a repetition percentage higher than 60%. However, it is in line with the fact that we almost reached the advertised size of its pool (4M). OL and SM percentage are similar and set to around 50%. In this case, since the claimed pool sizes are much bigger (100M and 40M respectively), we would have expected a lower percentage of repetitions.

The second last column of Table 7.1 displays the average and the standard deviation percentages of repeated IPs per server. In the last column of the table, we can see the same statistics per source client. These percentages are much smaller than the ones obtained considering all machines. Furthermore, the standard deviation is low, telling us that the frequency of IP repetitions is stable for each machine. We can see the biggest variation for BD. This is due to the company stopping our subscription just after the introduction of new machines. Because of this, the traffic in these machines and the repetition rate are lower and it influences the standard deviation.

The percentage of repetitions per path is even smaller. The obtained values are 0.2±0.1%, 1.5±0.3%, 6%±0.3%, 1.3±0.1% for, respectively, BD, OL, PR and SM.

This data, shows, once again, that RESIP providers, try to minimize the usage of the same GATEWAY IP for a single path. This minimization protocol is an incentive for the clients of RESIP services to use only one machine to send out many requests to one (or more) server(s) they want to contact. In



(a) BD

(b) OL

(c) PR

(d) SM

Figure 7.2: Hilbert curves of the GATEWAY IPs distribution of RESIP providers in RTT_DS.

this way, they maximize the IPs used for their requests and complicate the detection on the server side.

7.2.2 F2: Non Correlation between GATEWAY IP and Destination Server Locations

In our setup, we have two machines per location. If RESIPs chose the GATEWAY to minimize the additional distance between client and server introduced using their infrastructure, we would have higher percentages of repetitions of GATEWAY IPs between servers in the same location than among other ones. Indeed RESIPs would choose GATEWAYs close to the two machines more often, increasing the probabilities of repetitions and diminishing the pool size of the provider seen by those servers.

However, examining the unique IPs per provider per couple of servers, we see that the percentage of repetitions is similar in all combinations. Furthermore, the proximity study proposed in Section 4.3.4 tells us that the number of connections where client, server and RESIP machines are close to each other is low. This information suggests that RESIP providers do not choose a subset of GATEWAYs close to the destination server location.

7.2.3 F3: Non Uniform Distributions of GATEWAYs IPs

Figure 7.2 shows the Hilbert curves [129], a continuous fractal space-filling curve, for the GATEWAY IP addresses of every studied provider. We created the curves with ipv4-heatmap [130]. Each pixel represents a single /24 block. The color of each pixel depends on the number of addresses of that block that we collected during the experiment. A white pixel means that none of our IP addresses is in that /24 block. Colored pixels tell how many IPs are in the block. Colors range from blue (1 IP) to red (256 IPs).

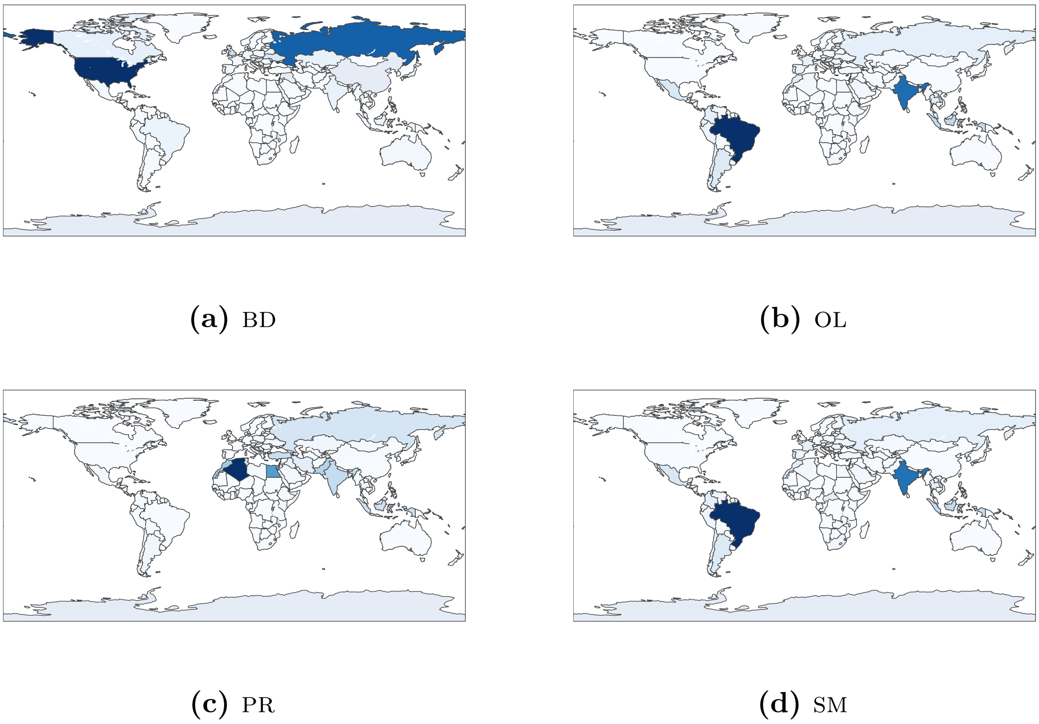
The curves are annotated with IANA labels. We can see in the top right, the Multicast, and Reserved blocks. On the top left, we can notice the blocks that were assigned to private companies before Regional Internet Registries (RIRs) became in charge of IPv4 allocation. The majority of the rest of the blocks are divided among the world's five biggest RIRs: American Registry for Internet Numbers (ARIN) [131], Réseaux IP Européens[25] (RIPE) [69], Latin America and Caribbean Network Information Centre (LACNIC) [132], Asia Pacific Network Information Centre (APNIC) [133], African Network Information Centre (AFRINIC) [134].

We can clearly see that the majority of BD GATEWAY IPs are situated in the areas controlled by ARIN and RIPE. On the other hand, the distribution of PR GATEWAY IPs have peaks in AFRINIC. Most of the GATEWAYs of OL and SM are in LACNIC and RIPE registries.

We can look at the distribution of GATEWAYs also from a geographic angle. Fig. 7.3 shows the distribution in percentage of GATEWAY IPs per country for each provider. The darker the color blue is in a country, the higher the percentage of GATEWAY IPs from that country. We see that BD requests mostly originated from the USA and Russia, while PR ones were sent mostly from North Africa. OL and SM larger fractions of IPs are situated in Brazil and India. These two providers exhibit similarities in terms of the respective geographic distribution and Hilbert curves.

These data show that single providers do not have uniform distributions of their GATEWAY IPs around the world. Furthermore, it shows how SM and OL distributions are comparable, while the ones of PR and BD differ from each other and from the two previously mentioned ones.

[25] French for "European IP Network".



(a) BD

(b) OL

(c) PR

(d) SM

Figure 7.3: Distribution of the gateway IPs per country.

7.2.4 F4: Different Management of superproxy IPs among Providers

When the client needs to send a request, it identifies the superproxy through a domain name. Analyzing the superproxy IPs obtained from the Domain Name System (DNS), we can see that PR superproxy is identified with the same 2 IP addresses from all locations. OL domain name resolution results in 18 IP addresses. 17 of them are shared among all our clients. One IP is observed only by both our machines in India and Canada and one among the two we have in India, Australia, and Tokyo. For SM, the domain name is resolved with 11 different IPs, seen by all our clients.

BD presents a different scenario. The total number of IPs is 5,603. For each machine, the resolution gives 3,539 IPs on average. The maximum number of shared IP addresses is 4,177. We have checked if BD is using its gateways network to play the role of superproxy as well. The data of our experiment reject this hypothesis since there is no intersection between the gateways and superproxys sets of IPs of BD.

From these results, we can see that BD has a more distributed network of

Table 7.3: Shared IPs among providers (RTT_DS).

	BD	OL	PR	SM
BD	-	9%	5%	9%
OL	2%	-	8%	63%
PR	2%	13%	-	13%
SM	2%	61%	7%	-

SUPERPROXYs, while the other providers have a small number of addresses for this component. This lets us think that the infrastructure of OL, PR and SM is more similar with respect to the one of BD.

7.2.5 F5: OL and SM most likely Share Part (or all) of their Pools of GATEWAY IP Addresses

Table 7.3 shows, for each couple of providers, the percentage of IP addresses shared by the two. Each cell represents the amount of IP addresses shared by the provider of the line and the one of the columns with respect to the amount of unique IP addresses of the provider of the line. PR shares more than 10% of its IP addresses with OL and SM. Moreover, OL and SM share more than half of their pool.

It is possible that the software of different RESIP services is installed on the same device and that, thus, the same IP address ends up being part of more than one RESIP pool. It is also possible that distinct devices behind a NAT can run the RESIP software of different providers. In this case, the providers share the same (NATed) IP address. Moreover, as previously said, the same IP can be used by distinct devices on different days. However, it is highly unlikely that these scenarios account for more than half of the IP addresses of these two providers. Our intuition is that SM and OL share a significant part of (or all) their pools of addresses, be it knowingly or not.

It would not be the first time that two providers shared a consistent part of their IPs. In the first RESIP study [38], the authors display how two other providers (Geosurf and Proxies Online) have major intersections of their pools. Moreover, in [45], the authors discover a high correlation among the pools of three Chinese RESIP providers. Thanks to further investigations, they disclosed that the three services are controlled by the same underlying operator.

Nowadays, there are more and more RESIP providers on the market claiming to have access to tens of millions of residential IPs. The above-mentioned data shows that even if these sizes were real, some pools of IPs

Table 7.4: Distribution of GATEWAYs with respect to the initial TTL value and the associated OSes (RTT_DS).

RESIP	Linux, Ubuntu, Android, MAC OS X, iOS, Solaris, openBSD, Debian (TTL = 64)	Windows, Windows Phone, Android (TTL = 128)	BlackBerry (TTL = 255)
OL	97.23%	0.51%	2.26%
PR	6.33%	92.79%	0.88%
SM	97.27%	0.51%	2.22%

are not uniquely used by a single provider. If this is true on a large scale, the global pool of RESIP IPs is much smaller than the sum of the claimed numbers. This would be significant from a detection point of view. It would dramatically decrease the number of IPs to possibly detect and it would open the door to the use of blocklisting detection techniques.

7.2.6 F6: GATEWAYs of Different Providers support Different OSes

As explained in Section 4.2.4, we collected the Time-To-Live (TTL) corresponding to the first client TCP packet of each connection in RTT_DS. The TTL is a value initially set by the OS kernel and included in every network layer packet. This value is decremented by one per each network element that it crosses on its path between sender and receiver. It was designed to prevent endless routing.

This value has been widely used to perform passive fingerprinting in combination with other parameters [109], [135]–[137]. In this context, we use it to study the distribution on sets of different OSes of the RESIP GATEWAYs of different providers. This is the only information that the TTL alone can give us.

To reconstruct the original TTL set at the sender, we round the value observed in a connection to the next higher power of two, as suggested by Lippman et al. [138], starting from 64. Table 7.4, shows the distribution of the original TTL of the three studied providers for 14 days of the data in RTT_DS (14/04/2022-28/04/2022). Each initial TTL value is associated with

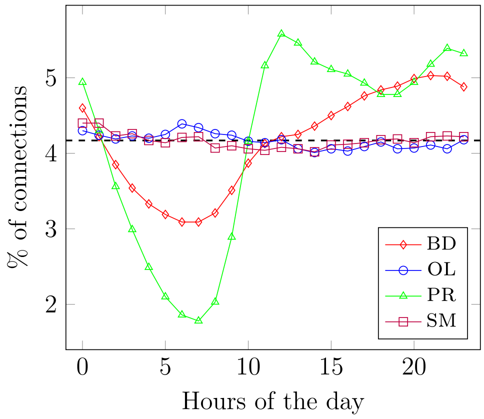


Figure 7.4: Diurnal patterns of GATEWAY IPs connections.

the OSes found in FINGERPRINTS_DS[26] for that value.

We can see that, for all providers, the usage of BlackBerry OS (TTL = 255) is very limited. On the other hand, the distribution among the other TTL values differs among the three. Most of the GATEWAYs of PR have an initial TTL of 128, while the vast majority of GATEWAYs of OL and SM have the TTL set to 64. This shows that OL and SM providers have access to similar categories of devices while PR takes advantage of different types of machines.

7.2.7 F7: Diurnal Patterns in the GATEWAY IPs Availability depend on Provider

For each received connection, we use the GEOLOCALIZATION_DS to localize the GATEWAY IP that sends the request. Combining this information with the CLIENT_EPOCH (UTC+0) we can determine the time zone in which the GATEWAY operates and the corresponding local time of the request.

In Fig. 7.4 we can see the distribution of the number of requests with respect to the local time. On the x-axis, we have the hour of the day, and on the y-axis the percentage of the number of connections sent by the GATEWAY in that local time, for each RESIP provider. A dashed line represents 4.17%, which should be the value of the percentage if the connections are equally distributed over the 24 hours. We can see that OL and SM present values around the dashed line, with a smaller prevalence of usage during the first hours of the day. On the other hand, BD and especially PR show to use devices as GATEWAY much more frequently in the second half of the day with

[26] We consider only the measurements with 100% confidence; we are aware that Android appears with 2 distinct values.

respect to the first half. This second daily trend has been also found in [38] for all the studied RESIP GATEWAYs.

This shows that not all RESIP providers use the same strategies. Some of them use devices available at any time of the day. Others, instead, take advantage of different classes of devices whose availability has a different diurnal pattern (e.g. mobile phones could be turned off at night while desktop computers could remain available).

7.3 Assessment of RESIP Pool Sizes

RESIP providers claim to have access to tens of millions of residential IPs and to be able to rotate them among the different requests of each client. For these reasons, the report [10] asserts that IP blocklisting has become "wholly ineffective". Doubtlessly, millions of different IPs cannot be blocklisted altogether and websites cannot risk blocking requests coming from real customers. As introduced in Section 6.3.2, if RESIP providers had access to these vast networks of IPs, the probability of seeing the same IP more than once during the collection would be low. However, the honeypot IPs, which our analyses suggest being RESIP IPs, showed a high percentage of repetitions. This is incompatible with randomly picking IPs from a pool of tens of millions IP addresses. Knowing the real order of magnitude of the RESIP IP pool can drastically change how we perform detection. Thus, we built a mathematical model to find this value from the IPs used by GATEWAYs.

Contrary to datacenter IPs, residential ones are dynamically assigned by Internet Service Providers (ISPs), that typically reassign IP addresses [139]. Thus, a user proxying out a request through the same device could end up sending requests with different IPs at different moments in time. Moreover, multiple devices could have the same IP at different moments in time or use the same IP at the same time when behind a Network Address Translation (NAT) device. For these reasons, there is no 1-1 correspondence between the number of devices and the number of IPs involved in a RESIP network. Hence, our analyses do not tell us the number of devices available to a RESIP provider at any point in time. However, counting the number of distinct IPs in a window of time, as we do, is usually considered to lead to an overestimation of the number of hosts [140]. Thus, our analysis likely provides an upper bound of this value. Finally, the goal of our analyses is not to find the accurate number of the IPs but to understand their correct order of magnitude. This can help us design effective detection methods.

In [C7], [J1], we applied a mathematical model[27] on the honeypot IPs. We

[27] To be exact, we applied two different mathematical models. The first one is based on

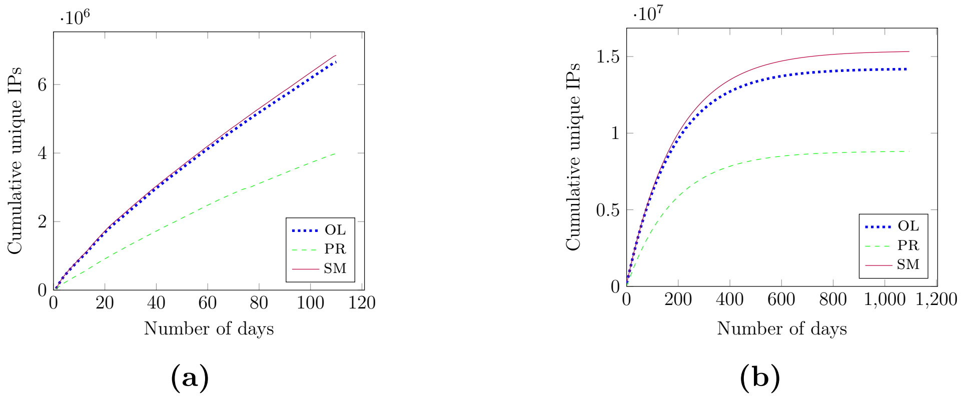


Figure 7.5: Cumulative curve (a) and projection in time (b) of new unique GATEWAY IPs.

saw that the pool from which these IPs were taken was likely in the order of the low tens of thousands. This result clashes with what RESIP services advertise. However, the IP dataset collected at that time was small and focused on a specific campaign leveraging a hypothetical RESIP provider. In this section, we apply the same model on the much larger RESIP IP dataset in RTT_DS to assess if our previous findings hold.

Hereafter, we present the model and the results obtained running it with the RTT_DS IPs[28] as input. Moreover, we discuss the obtained results and compare them to the ones obtained with the honeypot IPs.

7.3.1 Pool Estimation thanks to Cumulative Curve Fitting

In this section, we estimate the size of the RESIP pool by fitting the cumulative curves of new unique GATEWAY IPs in RTT_DS.

For each day, we consider the amount of different IP addresses that did requests on that day and were not seen before. We add them to the amount of unique IP addresses seen since the beginning of the experiment. Fig. 7.5a shows the result of this operation for OL, PR and SM. We can see that the

modeling the RESIP IP addresses seen at the server through cumulative curves. The second one models the process RESIPs do when deciding which GATEWAY and, thus IP address, is used for each new request, through different distributions. Both of the two approaches gave us similar results. For the sake of conciseness, we propose here only the one based on cumulative curves and its results. The interested reader can find more information about the second model in [C7], [J1].

[28] We did not apply the model on the IPs of BD because of the shorter collection period for this provider.

daily increment decreases over time, suggesting that it will eventually reach a maximum.

To confirm our visual understanding, we can fit the cumulative curves of new unique GATEWAY IPs with different curves and choose the one that best represents the phenomenon. Thanks to the Python library SciPy [141], we can perform this operation and find, for each curve, the parameters that give us the best Pearson correlation factor [142]. This factor ranges from -1.0, which means total negative line correlation, to 1.0, representing total positive line correlation.

We experimented with linear growth and decay as well as exponential growth and decay curves. Among these curves, the one that gave us the best Pearson correlation factor was the exponentially decaying curve ($a * (1 - e^{-(x-b)/c})$). We choose the parameters that give us a Pearson correlation factor of 1.0. We project the curves in time to find the value of the plateau.

This approach, most likely, does not enable us to obtain the exact number of IPs used by RESIPs in time. However, RTT_DS is a representative sample of RESIP connections for each studied provider, as explained in Section 7.1. Hence, we expect it to representatively characterize how transient the IPs available to RESIP are. We assume that this transient remains constant over time. Thus, by projecting the current data in the future we can have an idea of the correct order of magnitude of the pool size.

Fig. 7.5b shows the results of the projections. The curve of PR indicates that the plateau for this provider reaches values above 8M. This contrasts with the information given by the provider, which advertises a pool of half this size (4M). This could mean that they have more IPs at their disposal but they do not have them all available at the same time. Alternatively, they could count only one single IP per device they can use, not considering that the device can change its address. OL and SM plateaus are around 14M and 15M respectively. These values largely differ from the available information about the size of the pools (100M for OL and 40M for SM). Considering once more the fact that there is no 1 to 1 correlation between devices and IPs, the providers seem to overestimate the number of IPs they have at their disposal at any moment in time.

The obtained values do not validate the pool size obtained in [C7], [J1] where we applied the model on the honeypot IPs. There, we estimated RESIP providers to have a pool of IPs in the low tens of thousands. However, in our previous work, we applied the technique on a server in a unique location targeted by a specific scraper that could have used a different RESIP provider than any of the four studied in the experiment. In this case, instead, we are considering IP addresses from different locations. To perform a more fair confrontation, we can perform the same analysis also for every server of our infrastructure. Results show that for each server the plateau is bigger than 1M. In particular, we have values per server of 2,604,031±110,460, 1,818,972±33,534 and 2,737,814±145,454 for, respectively, OL, PR and SM. These results confirm, again, that the pool sizes of RESIP providers differ from what they advertise on the Internet. Moreover, this tells us that, if the honeypot IPs were RESIP IPs as hypothesized, they most likely belonged to a RESIP provider different from the analyzed ones and with a smaller pool of IPs at its disposal.

7.4 Lessons Learned Analyzing RESIP Connections

In the previous sections and chapters, we have looked at the RESIP ecosystem from different angles and we have discovered new features about their modus operandi. In this section, we present the lessons learned considering all our results together.

We have seen that all RESIP providers share part of their implementation and functioning strategy. RESIP services try to maximize the number of IPs associated with a single client-server path and do not seek to choose GATEWAY IPs close to the contacted server (Section 7.2.1 and Section 7.2.2). Moreover, the three providers studied in the Chapter 5, act in the same way when the server does not acknowledge packets. Indeed, all the RESIP services, try to contact the server from another couple of GATEWAYs located in different areas of the world (Section 5.3). Finally, as described in Chapter 4, none of the providers breaks the TLS session in between client and server, but they do so for the TCP session.

However, even if we see these common features, we have many indications that some of the studied parties have fundamental differences among themselves. Section 7.2.3, Section 7.2.6 and Section 7.2.7 show that the providers recruit GATEWAYs with different characteristics (installed OS, availability during the day) and from different areas of the world. In Section 7.2.4, we can see that BD has a completely different management of their SUPER-PROXY with respect to the other providers. All RESIP providers break the TCP and not the TLS session, but as visible in Figure 4.4, the difference between the RTT$_{TLS}$ and the RTT$_{TCP}$ produces different distributions for different providers. Finally, Section 5.3 shows that while all the providers use more than one GATEWAY to send SYN packets to non-acknowledging servers, PR keeps retransmitting packets from one port but OL and SM send SYN packets from newly opened ports. These results tell us that even if the providers show similarities, we cannot consider them as one monolithic entity. This complicates the task of finding detection methods able to identify all of them. At the same time, it opens the door for attribution techniques thanks to the detection of specific features of providers, as the one proposed in Section 5.4.

Among the different tests we propose, we can notice that OL and SM have high similarities in almost all the results. These two services share more than half of their GATEWAY IPs (Section 7.2.5) and they show the same distribution of /24 (Section 7.2.3) and OSes of their GATEWAYs (Section 7.2.6). Furthermore, neither of them shows a diurnal pattern in the availability of devices (Section 7.2.7) and they present similar distribution of the differences between TLS and TCP RTTs (Section 4.3) as well as the speeds used to send their packets (Section 4.3.1). Moreover, both of them use a very similar algorithm to send new SYN packets in case they do not obtain a SYN-ACK packet from the server (Section 5.3). In this last scenario, however, we see that they have shown a small difference. The median interval between the first and second GATEWAYs that contact the server is really short for SM (<1s) but not for OL (7s). While we can see striking similarities between these two parties, this difference and the fact that they have different SUPERPROXY IPs (there is no intersection among the two groups of IPs), does not enable us to say that these two companies are two different names for the same entity. The root cause of these similarities remains an open question.

Finally, we could implement an approach similar to blocklists to detect RESIP . We have seen that two out of three providers appear to have pool sizes much smaller than what they claim (Section 7.3.1). Moreover, Sections 7.2.3 and 7.2.5 tell us that two providers use the same /24s and that different providers share a high percentage of IPs. We need to remind that an IP registered on two different days, does not necessarily mean that the same device was part of a RESIP for the whole time between the two days. However, considering the corresponding /24 enables us to take into account devices that change address but remain in the same /24.

Based on this, our idea would be to build lists of suspicious /24. When an IP is seen as part of a RESIP, we mark the corresponding /24 as suspicious. When an IP from a suspicious /24 contacts a server, the server can answer with a more invasive technique, such as delaying the sending of the SYN-ACK packet (Section 3.5). Since we have seen that IPs are much more often reused for different paths and servers (Section 7.2.1), a collaborative protocol among different servers to build and share this list would enable to have better detection.

This approach might not always be effective. As shown by Griffioen et al. [143], there is no general rule among ASes to maintain the previous prefix after reallocation. While some ASes reallocate only in the same /24 for operator policy, others present a more variegated scenario. Because of this, our idea could not work and a real-world measurement is necessary to assess the validity of the proposed approach. We leave this as future work for this thesis.

7.5 Summary

In this chapter, we provide novel insights about the RESIP ecosystem thanks to the analysis of the RTT_DS. We show different aspects of the RESIP infrastructures, highlighting similarities and differences of the providers in the geographic distribution, types, management and amount of their GATEWAYs, as well as in the management of their SUPERPROXYs. Moreover, we estimate the size of residential IPs that are at the disposal of RESIP providers. Our results do not confirm the claims RESIP providers do on their websites. Finally, we use all the information gathered about RESIPs in this chapter and the previous ones to list all the lessons we learned about these parties. This information gives us indications about the next directions the security industry could follow to better detect and mitigate scraping requests through RESIP services in the future. However, while it is important to detect and mitigate RESIP requests enabling malicious activities, the original client behind these infrastructures is the actual party that we would like to detect and mitigate. Hence, it is important to acquire information about it. In the next chapter, we focus on this task, investigating the possibility of geolocating scrapers sending requests through RESIPs.

Chapter 8

Geolocation of Scrapers behind a Residential IP Proxy

8.1 Introduction

In the previous chapters, we saw how much and why scrapers leverage RESIPs to perform web scraping (Sections 2.2.3 and 2.3). Moreover, we presented and tested two new ways to detect server-side if a request is proxied through RESIPs (Chapters 3 to 5) and we characterized the RESIP ecosystem (Chapter 7). However, RESIPs are just instruments in the hands of scrapers and other malevolent parties that use them for their activities. While it is important to detect and mitigate RESIP requests when they enable malicious activities, the original client is the actual party on which we want to acquire threat intelligence information.

In particular, we would like to understand when different campaigns (different sets of requests proxied through RESIPs) are initiated in the same area of the world and, hence, could be produced by the same party. Moreover, we would like to see if multiple parties from different locations in the world are behind a single set of RESIP requests that reach a server at the same time. This information would help us to better characterize the goals and means of attackers, and, thus, defeat them. Within the current state-of-the-art, there is no available technique to perform these tasks.

In this chapter, we investigate the possibility of performing them with a new multilateration approach. In Section 2.5, we studied geolocation through multilateration. Moreover, we saw why we cannot leverage previous works on RTT-based geolocation in this new scenario. However, we can take advantage of part of their results. In [C8][29], we studied the problem of RTT-based

[29] For conciseness, in the rest of the thesis we refer to this preliminary work only

geolocation between directly connected machines. Our outcomes told us that considering a combination of speeds helps in the geolocation accuracy. Indeed, a single average mean value for packets traveling around the world does not exist, as discussed in Section 4.3.1 regarding RESIP connections in RTT_DS. Different speeds enable us to cover multiple scenarios and associate the best speed to each connection.

This concept is the starting point of our approach. As highlighted in Section 8.2, though, the problem at hand presents many challenges with respect to the direct connections studied in [C8]. Thus, we start our investigation by verifying *if* there is a combination of speeds that enables to geolocate the client in this complex scenario.

To answer this question, we enlarge and refine the technique in [C8] to obtain *RTTlocator*, our geolocation algorithm for the initiators of RESIP requests. *RTTlocator* leverages the δ_{RTT} values of a set of RESIP requests to multilaterate the location of the original client using a combination of speeds. Our experimental results on ground truth connections show that the algorithm geolocates the original client when its parameters are optimized for the specific set of requests. However, finding *the optimal* combination of speeds that reach an accurate solution for all sets of requests considered together and/or obtaining multiple client locations is not a trivial task. We show that, although intuition would say otherwise, considering the best speeds that reduce the median error and its dispersion among different sets of requests is not an effective approach.

While finding the optimal combination of speeds remains an open problem, we present here our early results and the knowledge gained in studying this complex problem with *RTTlocator*. We believe this to be an important starting contribution to finding the location of the client(s) behind RESIPs.

In the following, we detail the idea behind our technique and the challenges imposed by the setup (Section 8.2). Then, Section 8.3 describes the algorithms used in *RTTlocator*. Section 8.4 proposes an evaluation of the *RTTlocator* on the ground truth dataset obtained from the RTT_DS (Section 4.2.4). The section explains the different used parameters and the problems in choosing the right set of those. Moreover, it provides a technique to find a possible final *Set of Speeds* and the accuracy obtained using them. Furthermore, it proposes the geolocation of real-world RESIP connections obtained through our collaboration with Amadeus IT Group. In these requests the original client location is unknown. Finally, Section 8.5 discusses the results and limitations of the technique.

highlighting the parts that are important to understand our new algorithm. We refer the interested reader to the full paper for more information.

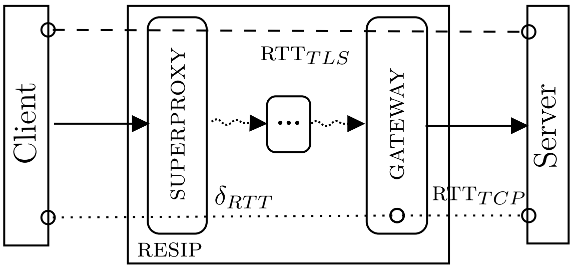


Figure 8.1: Common internal structure of a RESIP provider and RTT values among the involved parties.

The content of this chapter is based on ongoing work. The high-level idea behind multilaterating the original client behind a RESIP was introduced as future work in [C3]. The author contributed to [C8], where the geolocation algorithm we leverage and refine in this chapter was first applied to a different problem.

8.2 Multilateration of the Original Client

Figure 8.1 recalls the common internal structure of RESIPs as discussed in Section 3.2. The client sends the request to the SUPERPROXY, the request traverses the RESIP infrastructure, composed of an unknown number of intermediate machines, if any [38], and reaches the GATEWAY. This last element contacts the target server. In this scenario, the TCP and TLS sessions take place between different parties (respectively GATEWAY-server and original client-server). We can identify this difference in the transport layer at the server-side thanks to the comparison of the RTT of the TCP and TLS packets (Section 3.4).

The difference between these two RTT values (δ_{RTT} =RTT$_{TLS}$-RTT$_{TCP}$) reveals information about the distance between the used GATEWAY and the original client. We can exploit this information, given a set of requests initiated by the same client and using multiple different GATEWAYs, to multilaterate the location of the client. We can find the intersection of the circles whose centers are the GATEWAY locations, information we can retrieve from their IP address, and whose radii are half of the corresponding δ_{RTT} multiplied by the speed of the packets. This intersection identifies the area of the world where the original client is. Figure 8.2 provides a visual example of this concept.

However, accurately performing this process comes with a number of complex challenges. The machine we want to geolocate is hidden behind a set of other machines (RESIP infrastructure). On the server side, we do not have

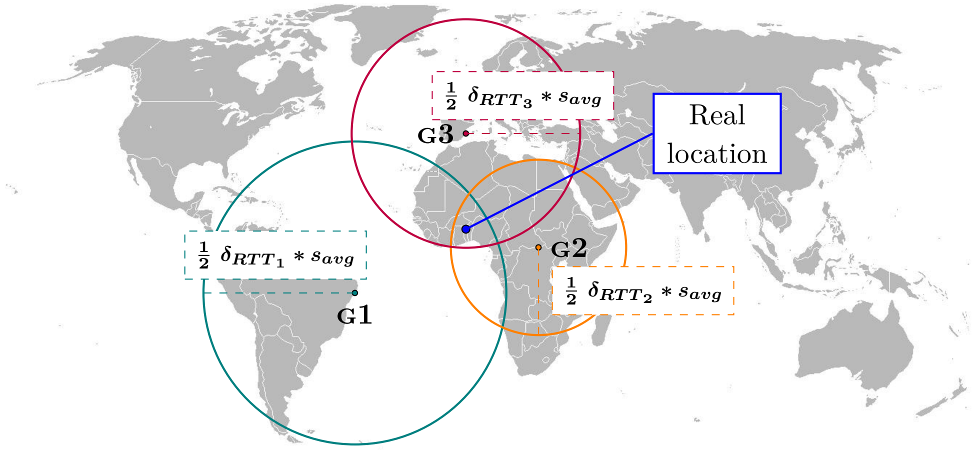

$\frac{1}{2}\ \delta_{RTT_3} * s_{avg}$
G3
Real location
$\frac{1}{2}\ \delta_{RTT_1} * s_{avg}$
G2
$\frac{1}{2}\ \delta_{RTT_2} * s_{avg}$
G1

Figure 8.2: Example of multilateration from 3 GATEWAYs using the corresponding δ_{RTT} values.

any information, e.g. number of machines and their location, about these machines. The only information we know is the δ_{RTT}, which is not measured directly. It is obtained from the difference between the measured RTT$_{TLS}$ and RTT$_{TCP}$.

Moreover, this metric is not the RTT between two machines but is the sum of the RTTs among the unknown number of machines that break the TCP session between the GATEWAY and the client. Furthermore, the machines in the RESIP infrastructure could produce large detours of packets (e.g. the client and the GATEWAY are in Europe, but the SUPERPROXY is in South America) that are reflected in the value of the δ_{RTT} and bias the resulting information about the distance between the client and the GATEWAY.

In addition, on the server side, we do not have any control over the RESIP and their choice of GATEWAYs. Hence, we usually have one single δ_{RTT} measurement per GATEWAY and no information if that measurement had delays due to network congestion.

Finally, even if the δ_{RTT} would give us the precise time to go from the client to the GATEWAY and come back, there is no common value for the speed of packets traveling around the world (Section 4.3.1) and the different journeys between machines inside the RESIP could experience different packet speeds.

To address these challenges and overcome the limitations of previous RTT-based multilateration methods in our use case (Section 2.5), we propose a probabilistic approach to geolocate the initiator(s) of a set of requests sent through RESIP services. Starting from the algorithm in [C8], we create a new technique, called *RTTlocator*, that multilaterates the client behind a

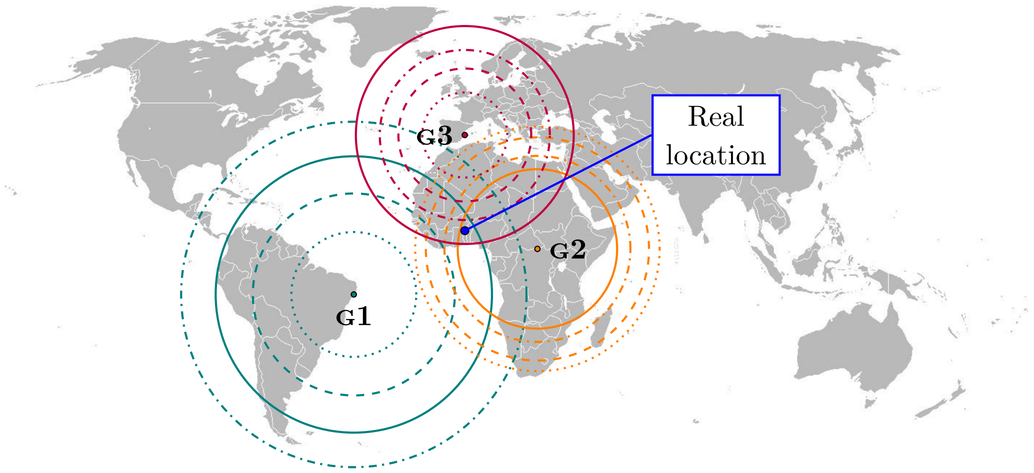


Figure 8.3: Multilateration using multiple average packet speeds. Each circle corresponds to a different average packet speed. Solid lines are the ones that best contribute to finding the solution area in this example.

RESIP leveraging the δ_{RTT} and considering multiple packet speeds. For each connection coming from a GATEWAY, we end up with a set of circles centered in the GATEWAY location with different radii, one for each considered speed (Figure 8.3). We expect the circle that best represents the real distance between the client and the GATEWAY to cross in the same area as the ones of the other GATEWAYs that best represent the corresponding real distances.

We acknowledge that there are reasons why neither of the chosen speeds could end up properly modeling the packet path between the client and the GATEWAY. Network delays and large detours of packets in the RESIP infrastructure could largely affect the δ_{RTT}. There could be really different packet speeds among some machines of the RESIP infrastructure that would bias the result.

In these cases, which we hope to be a minority, none of the concentric circles obtained with different speeds and centered in the GATEWAY location cross on the location of the original client. Moreover, they add noise to our geolocation process. For this reason, in our algorithm, we first filter out the circles that do not contribute to finding the solution because they do not cross in a single location with the majority of the others. Then, we just consider the crossing circles to understand the area of the world in which the client is likely to be.

Beyond finding the location of the original client, *RTTlocator* has the potential to tell if multiple clients are behind a set of requests. If two clients in different parts of the world send requests using RESIPs at the same time,

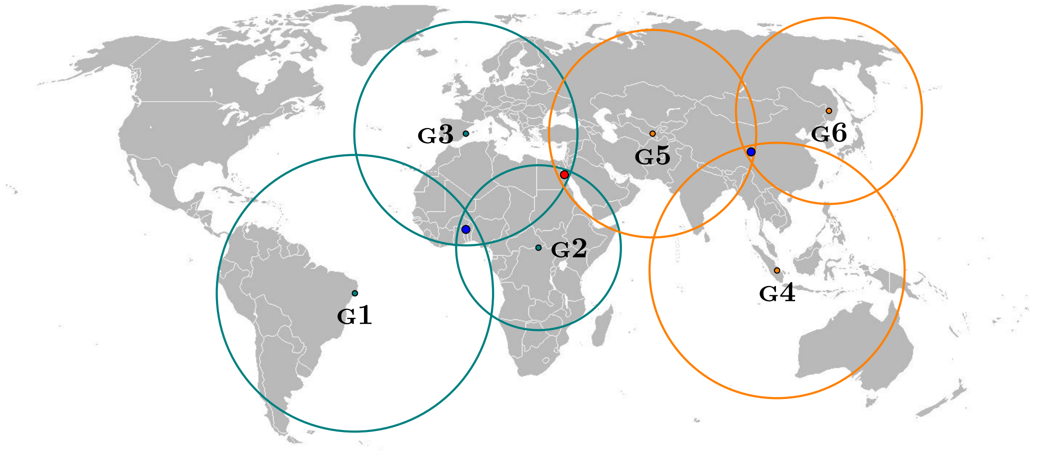


Figure 8.4: Example of two scraping campaigns initiated by different clients and running at the same time. The first client leverages G1-G3, the second G4-G6. Blue dots represent the real client position and the red one is the erroneous cross of circles from GATEWAYs of different campaigns.

we obtain two groups of circles intersecting among them in those areas.

However, the intersections across the two groups will be all over the world. To identify these groups of circles, we need to detect when the majority of circles cross at just one point or more than one. Moreover, we need to carefully tune the initial filtering of the circles to not remove the circles contributing to a second location. Finally, having multiple campaigns running at the same time could lead to wrong solutions. Figure 8.4 represents an example of this situation. The blue dots represent the two original clients. The first one leverages G1-G3 to send out requests, the second G4-G6. Circles obtained from G1-G3 cross the area of the corresponding client, circles created from G4-G6 meet in the second client location. However, circles from G2,G3 and G5 also all cross among them in correspondence with the red dot, which could be wrongly considered a solution. We need to have a procedure that does not lead to this last result.

The RTT*locator* performs the above-mentioned tasks leveraging two distinct algorithms:

- **Grid Based Localization (GBL)**: We consider the planisphere as a grid of squares. We build a circle from each RESIP connection and each speed to study. The circle is centered in the GATEWAY location and the radius mirrors the distance between the GATEWAY and the initiator, as expressed by the δ_{RTT}. We find the square(s) crossed by most circles. This gives us an initial idea of the area(s) of the world in

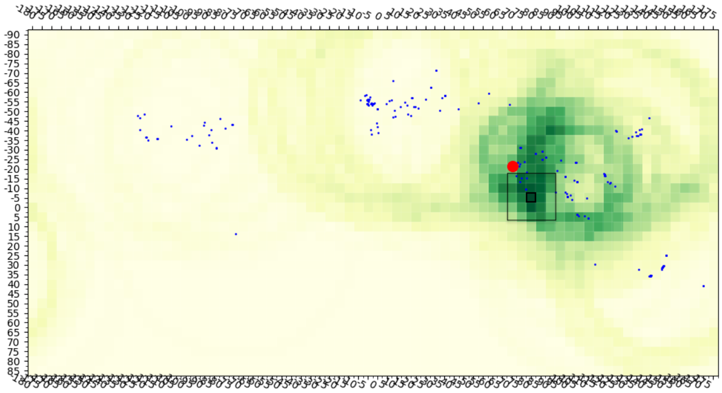


Figure 8.5: Example of visual outcome of the GBL algorithm.

which the client(s) is(are) located and identifies the circles with valuable information (the ones that contribute to estimating the most crossed square(s)) for the next phase. GBL enables us to filter out the circles that produce noise in the geolocation and to see if multiple clients are behind a set of requests.

- **Iterative Least Squares Multilateration (ILSM):** For each identified area of the world in which a client could be, we consider the circles passing in that area. We apply a refined version of the Iterative Least Squares Multilateration Algorithm in [C8] to find the location of the client.

In the next section, we explain step-by-step both the algorithms that compose RTT*locator*.

8.3 The RTT*locator* technique

Hereafter, we provide the details of two algorithms behind RTT*locator*, defining some *Parameters*. These parameters influence each other. To find the optimal combination of them, we take advantage of a ground truth database, in which we know the location of the clients behind RESIP requests. In Section 8.4, we present the chosen value for each parameter and explain the ratio behind these choices while, in the following, we just refer to the concept expressed by each parameter for an easier understanding of the logic of the algorithms.

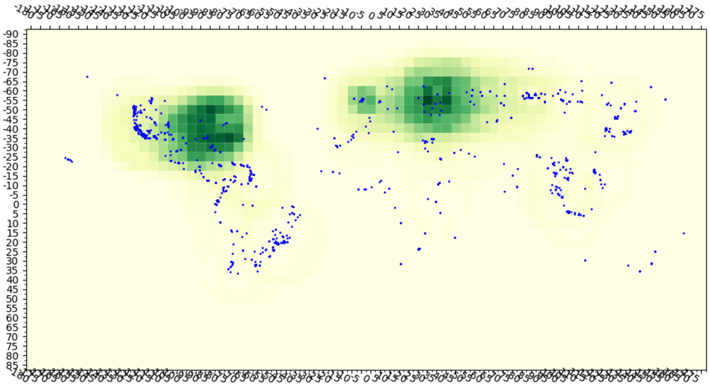


Figure 8.6: Example of multiple client locations as visual outcome of the GBL algorithm.

Grid Based Localization

We consider the planisphere centered on the European continent. We divide the planisphere into a grid of squares, whose length is *Grid Size*. Each square has a counter that is initialized at zero. We use this counter to know how many circles cross each square.

We read an input file that contains information regarding a list of RESIP connections. Each line of the file provides the latitude and longitude of the GATEWAY contacting a server as well as the corresponding δ_{RTT} of the connection expressed in milliseconds.

Initially, we confirm the existence of connections created by a minimum of three distinct GATEWAYs, each having non-matching coordinates. If this condition isn't met, the multilateration process becomes unfeasible, leading the algorithm to generate an error. Then, the algorithm verifies that the δ_{RTT} is in the interval [*Minimum* RTT, *Maximum* RTT]. This check enables us to eliminate errors in the measurement and long delays at the TCP or TLS layer due to network congestion. If this check is not passed, the line is not considered in the further steps of the algorithm.

At this point, we divide the δ_{RTT} value by 2, obtaining $\frac{\delta_{RTT}}{2}$. We perform this step since we only need the one-way time to go from the GATEWAY to the client. The entries remaining after the cleaning process are sorted in ascending order based on the $\frac{\delta_{RTT}}{2}$ value.

For each line, we draw the corresponding circles in our grid. We consider the latitude and longitude of the GATEWAY to find the center of the circles. We determine the radii in kilometers by multiplying $\frac{\delta_{RTT}}{2}$ by each considered speed in the *Set of Speeds*.

After this, we convert the coordinates from degrees and kilometers into square distances. We plot the circle on the planisphere. If the circle exits the planisphere on the east or west border (longitude -180° and +180°), we draw the exiting part of the circle on the other side of the planisphere. Indeed, we consider the planisphere as a cylinder where the line at longitudes -180° and +180° coincide. We thus draw the circle as if there was continuity between the two areas around the line. Regarding the circles that exit the map on the north and south borders (latitude -90° and +90°), we make the reasonable assumption that the path does not cross the poles. Hence, for these circles, we just consider the part not exiting the planisphere.

For each square that is traversed by a circle, we increment its counter by one. After processing all lines, the counter of each square indicates the total number of circles passing through it. Figure 8.5 provides a visual representation of an example result of this phase. It represents the planisphere divided into squares. The x and y axes provide respectively the longitude and latitude values. Blue points refer to the locations of the considered GATEWAYS, and the red dot corresponds to the real client location. Each square has a different color depending on the number of circles passing through it. The darker the color, the higher the number of circles passing through it. We can see that the area near the solution has the darkest squares, recognizing that the requests were sent from that part of the world.

When the grid shows multiple regions on the planisphere with darker squares (e.g. Figure 8.6), it may indicate the presence of different clients behind the set of connections. We start considering the square with the highest count. We retrieve the circles crossing that square. Moreover, we consider that also the circles crossing the neighboring squares could contribute to finding the location. We consider neighboring squares the 24 squares that surround the highest value square in the two preceding/subsequent rows and columns.

Figure 8.5 provides a visual example of the concerned squares that are highlighted with a black border. We retrieve the circles crossing those squares as well. Depending on the location of this group of squares in the world, we define which projection of the planisphere (Europe, America, Asia centered) the next phase of our algorithm (ILSM) needs to use. One of ILSM limitations is that it does not cope with those circles that exit the planisphere on the east and west borders. To minimize this phenomenon, we need to use the earth projection which positions the area under study as close to its center as possible.

We then calculate the number of retrieved circles over the amount of total plotted ones. If this percentage is lower than *Minimum Circles*, we consider that at least one other client location exists and revisit the results obtained with the GBL algorithm. We then take into account only the plotted but non-previously retrieved circles. We find a new square with the highest count and, as before, we retrieve the circles crossing it and its neighboring squares. This corresponds to a second location. We recalculate the number of retrieved circles over the amount of total plotted ones, considering also the previously retrieved ones. Suppose the percentage is still lower than *Minimum Circles*, we redo the process to identify a third location. This iterative process continues until the threshold percentage is reached.

For each recognized location, we obtain a set of circles that best enable us to geolocate the solution and the corresponding projection.

Iterative Least Squares Multilateration

In the previous phase, we extracted the most contributing circles for each client location. Each circle radius was calculated with the speed among the used *Set of Speeds* that best contributed to the solution. Moreover, for each client location, we determined the corresponding planisphere projection to use. Given this information, we take advantage of a refined version of the ILSM algorithm described in [C8] to find a better approximation of the location of each client.

In [C8], we showed how to use the RTT values associated with a set of packet speeds for geolocation purposes. The use case was much simpler though: we were finding the location of a server through the RTTs obtained from direct connections of clients. In our case, we want to localize the client behind the RESIP given the GATEWAYs and the δ_{RTT}. In this new scenario, the δ_{RTT} is a much worse estimate of the distance between the client and a GATEWAY because the packet traverses the RESIP infrastructure, composed of multiple machines, which could impose a large detour of packets.

In [C8], we ran the analytical solution with several random samples of circles to minimize errors caused by "erroneous" circles. Thanks to the GBL phase, we are not facing this problem and we can run the ILSM algorithm only once. This enables us to be much more efficient.

Moreover, we only perform multilateration on one projection for each set of connections, since we already know which projection to use from the previous step. In the original algorithm, three projections were used to understand which one was the one giving the best results. Finally, in the original algorithm, we were considering the number of hops passed by the incoming request to refine the geolocation. However, this information is not available in the proposed scenario and thus we do not perform this step.

Given a set of circles and the projection, the algorithm uses the Iterative List Square method [144] to find the point that minimizes the squared error between this point and all the circles. The algorithm returns the latitude and the longitude of the solution. It calculates the error between this point and the real client location (Haversine distance [111]) when this information is known.

8.4 Evaluation

In this section, we first define the datasets that we take advantage of to evaluate and tune *RTTlocator* (Section 8.4.1). Then, in Section 8.4.2 we use the ground truth dataset to tune the different parameters of the algorithm and try to understand their optimal value. Finally and with not less importance, we show the results of running *RTTlocator* on real-world scraping connections passing through RESIP services.

8.4.1 Datasets

Ground Truth Dataset

Our ground truth dataset is obtained from RTT_DS. Section 4.2.4 presents the collection process. For our analysis and evaluation, we take samples from this collection. We arbitrarily chose the size of the sample to be a manageable size for the running time of the algorithm. For each RESIP provider and each client, we retrieve 10,000 connections, divided into 2 groups of 5,000 connections. This corresponds approximately to 20 consecutive hours of connections per client and RESIP. We obtain a grand total of 4*22*10,000=880,000 connections.

For each sample, we know by design the location of the corresponding client and the used RESIP provider. For each connection in the sample, we know the latitude and the longitude of the GATEWAY, retrieved from the IP address thanks to MaxMind GeoLite2 Database [70] and the corresponding registered δ_{RTT}. We use this information as input for *RTTlocator*.

Real World Datasets

As explained in Section 4.4.2, we have a collaboration with Amadeus IT Group and the third-party company protecting their domains from bot attacks. Thanks to this, we have access to RESIP real-world connections sent to their domains. In these requests the original client location is unknown.

We focused on four different airline domains and we obtained the corresponding datasets, which we call *Airline X Dataset*, *Airline Y Dataset*, *Airline Z Dataset*, *Airline W Dataset*.

To extract these datasets, we asked the security analysts working in Amadeus IT Group to indicate already built fingerprints not including the

RTT_DETECTION technique and whose corresponding traffic could be proxied through RESIPs, based on their experience. As we saw in Section 4.4.2, the technique can outcome false positives when used alone. We decided to add it as a parameter of fingerprints only in a second step to reduce these cases. For *Airline X Dataset* and *Airline Y Dataset* we used the same bot signature, while we use different ones for the other two datasets.

To confirm the RESIP intuition of the analysts, we applied the RTT_DETECTION technique on the δ_{RTT} values of the connections matching these fingerprints. 99.6% of them had a δ_{RTT} higher than 50ms and thus corroborated the initial intuition of the analysts.

For each airline, we built a dataset of 10,000 connections, to be comparable with the size of the ground truth files. For each connection, we obtain the latitude and longitude of the IP address, thanks to the MaxMind GeoLite2 Database [70], as for the ground truth one, and the corresponding δ_{RTT}.

8.4.2 Parameters Tuning

In this section, we study the parameters introduced in the description of the algorithm. We take advantage of the ground truth dataset, in which the location of the original client is known, to find the most appropriate values for them, when feasible.

Minimum RTT

Minimum RTT is the value below which we consider the δ_{RTT} to be the cause of an error. δ_{RTT} is the difference between the RTT$_{TLS}$ and the RTT$_{TCP}$ and, as described in [C2], having this difference above 50ms is a clear indication of a RESIP request. However, some RESIP requests show a δ_{RTT} even below this threshold, which was obtained to minimize false positives Section 4.3.2. We chose to set the *Minimum RTT* to 25ms which is the first value after which the RESIP curves in Figure 4.5 start to show values above zero.

Maximum RTT

Intuitively, it seems reasonable to think that a longer packet journey is more likely to be error-prone than a short one. Indeed, the longer the path the higher the probability of having delays caused by the network. Moreover, RESIP infrastructures could cause large detours of packets in specific connections and we would like to filter them out of our processing. *Maximum RTT* is a threshold value above which the risk of an error is too high.

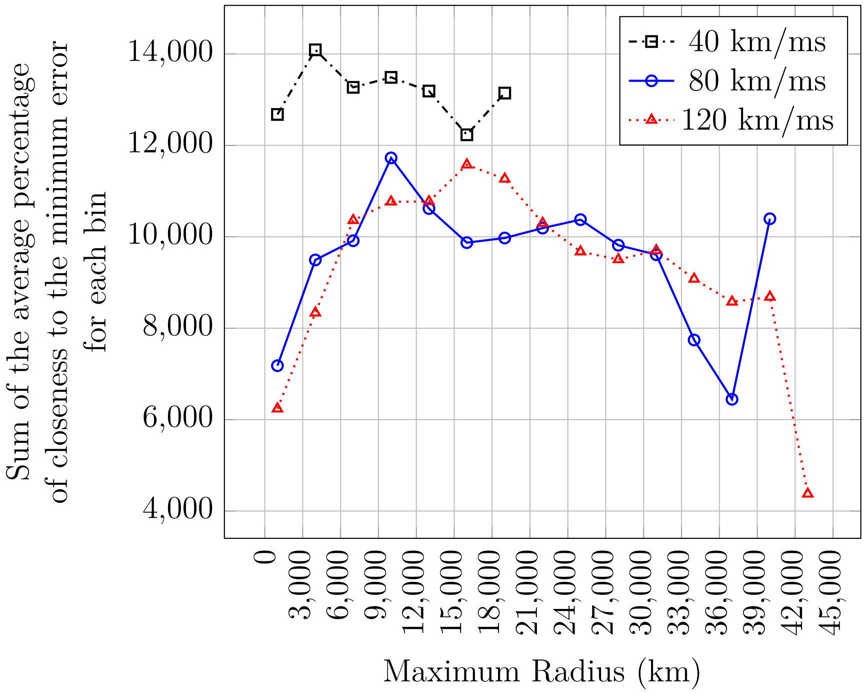


Figure 8.7: Maximum radius and corresponding percentage of closeness to the minimum error for different speeds, using the GBL algorithm standalone.

To find this value and understand if it changes in the function of the used speed, we run only the GBL logic of the algorithm. We find the square with the highest count and we calculate the error with the real solution using the center of that square. We use a *Grid Size* of 5[30] and multiple single speeds that enable us to represent low, medium and high-speed connections (40,80,120 km/ms).

For each RESIP provider, we consider the sample files with 5,000 connections and we order the entries in ascending order based on the δ_{RTT}. We run the GBL algorithm on each percentage of lines from 1% to 100% with step 1%. We call each of them a batch. For each batch, for each tested speed, we know which is the obtained maximum radius (speed$*\frac{\delta_{RTT}}{2}$). We calculate the lowest error for each batch. Thus, for each single file and each speed, we obtain 100 batches and we know for each of them the lowest error and the maximum radius in the batch.

For each file, for each speed, we find the minimum error among all batches. We divide the error of each batch by this value and we multiply it by 100. In this way, we know how close the error of a batch is to the minimum possible

[30] Size of the grid in the GBL algorithm.

Grid Size	**Total Time (s)**	**Median Error (km)**
1	1,816	7,0148.04
5	237	6,9812.11
10	159	7,0210.30
20	146	7,6857.35

Table 8.1: Total time and median error of the GBL algorithm standalone using different *Grid Size* values.

error for that file and speed.

Figure 8.7 provides a visual representation of the result. Each batch is represented through the corresponding maximum radius. We divide the maximum radius in bins of 3,000 km to obtain a more readable result. For each bin, for each RESIP provider, we find the average percentage of closeness to the possible minimum error among all files. For each bin, we sum the average percentage of the closeness of each provider to have a global view. We have one curve for each analyzed speed. We can see that each speed has the highest sum in different areas of the plot. In particular, for 40 km/ms we obtain the best values around 5,000km, for 80 km/ms around 10,000, and for 120 km/ms around 15,000.

These results suggest that the maximum radius is not a function of the studied speed. It is a constant was value is around 125 ($125 = \frac{5000}{40} = \frac{10000}{80} = \frac{15000}{120}$). Thus, we can consider a single value for the *Maximum RTT* to have a more efficient run of the algorithm and select the best circles to consider in the next phase. In particular, the *Maximum RTT* we consider for each speed is 250ms (e.g. 2*5,000km/40km/ms).

Grid Size

As explained in Section 8.3, GBL divides the planisphere into squares of size *Grid Size*. We want to find the value of this parameter that gives us the best compromise between accuracy in finding the location and the time taken to find that solution. We run all the files of each RESIP with the GBL algorithm with different values of the *Grid Size*: 1,5,10,20. The grid is indexed by latitude and longitude degrees. Hence, a *Grid Size* equal to 1 means that we obtain 1 square for every degree of the grid, for a total of 360 columns and 180 lines.

As for studying the *Maximum RTT*, we calculate the error with the center of the square with the highest count. We obtain the errors in finding the solution and the corresponding running time, for each *Grid Size*, file and tested speed (40,80,120 km/ms). For each *Grid Size*, used speed and RESIP, we find the average error and the total running time.

Table 8.1 represents the results. For each tested *Grid Size*, it displays the corresponding sum of the running time and the average error among all RESIPs. We can see that with the increase of the used *Grid Size* the running time decreases but the difference between running with *Grid Size* 1 and any other *Grid Size* is much higher than just considering the other grid sizes. Considering the average error, we see that *Grid Size* 5 gives us the best result. While we would have expected *Grid Size* 1 to have the lowest error, most likely the small size of the squares spread all the circles that contribute to the solution among different squares. Thus, the one with the highest count could be "far" from the solution. When considering grid size 5, the square includes all the circles of the previous 25 *Grid Size* 1 squares and has a much higher contribution in terms of circles. For *Grid Size* values higher than 5 we see that the error increases. Indeed, considering bigger and bigger squares we lose in accuracy to find the solution. For these reasons, we chose to run our algorithm with *Grid Size* 5.

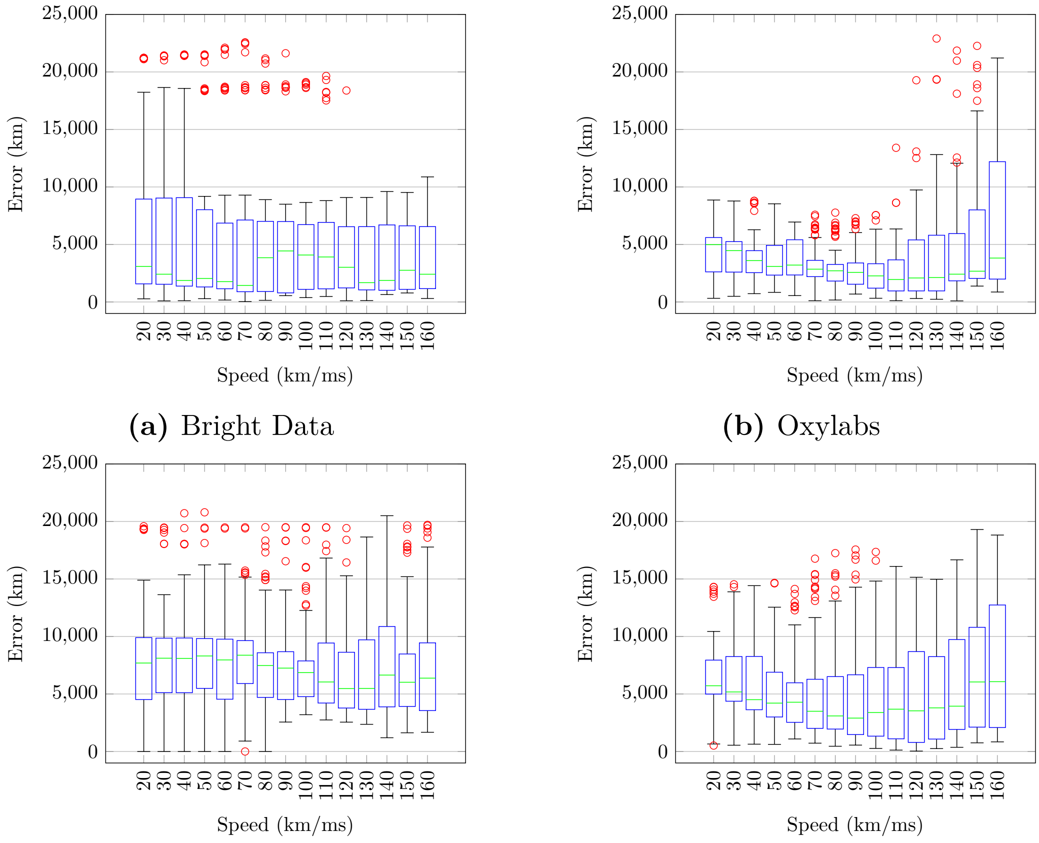
Set of Speeds

As explained in Section 8.3, we use a *Set of Speeds* to try to identify for each GATEWAY the most appropriate speed. This speed produces the circle that best represents the distance between this machine and the original client.

However, we need to find the components of the *Set of Speeds* that enable us to have the best accuracy for all the sample files. We initially try to find the combination that minimizes the error when only one client is behind a set of requests.

To understand which set to consider, we run the *RTTlocator* on the connections in each sample file, divided among different RESIP providers. We assume that each RESIP provider could have a different internal infrastructure and, for this reason, we consider the corresponding files independently. For the previously discussed parameters, we use the values chosen in the dedicated section. Thus, *Grid Size* is set to 5, *Minimum* RTT to 25ms and *Maximum* RTT to 250ms. For the *Minimum Circles*, we set it to zero so the GBL algorithm retrieves only one solution.

We test single speeds starting from 20km/ms to 160km/ms with a step of 10km/s. For each RESIP provider, we calculate the median error among the results obtained in each file. Moreover, we also consider the corresponding standard deviation and the interquartile range to measure the statistical dispersion. To find our final *Set of Speeds*, we do not want only to minimize



(a) Bright Data

(b) Oxylabs

(c) Proxyrack

(d) Smartproxy

Figure 8.8: Error in geolocating using *RTTlocator* with single speeds.

the error of the majority of samples but also to have a small dispersion.

Figure 8.8 provides a visualization of the results in running *RTTlocator* with single speeds. On the y-axis, we have the error in kilometers. On the x-axis, the tested speed. For each speed, a boxplot summarizes the errors on the files of the corresponding provider. The plot tells us that a single speed that provides a good result for all the considered files does not exist, as expected. Moreover, it seems to suggest that the different providers, as we suspected, have different internal infrastructures. The error for PR is much higher than the other proxies for all the considered speeds. Moreover, while for BD the median error is lower for high and slow speeds, OL and SM show an opposite pattern. We can also see that for every single speed, there are files for which the error has low values (lower part of the whisker of each boxplot). This information confirms to us that, when the right speed is chosen for a set of requests, the algorithm can find an accurate solution. However, we see

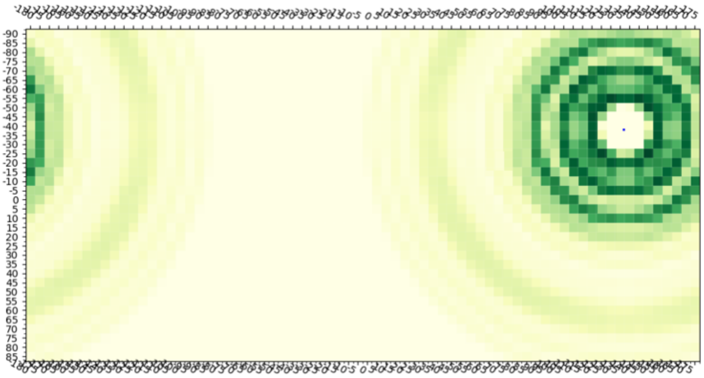


Figure 8.9: Outcome of the GBL algorithm for *Airline W Dataset*.

that other files obtain a much larger error for the same speed.

To try to balance the effects of each speed on each file, we would like to find the *Set of Speeds* that enables us to obtain low errors for the majority of files. At the same time, we would like to reduce the dispersion of the error.

To try to reach this goal, we create a point system. For each RESIP provider, we consider three metrics: median, standard deviation and the interquartile range (IQR). For each metric, we order the tested speed from the one resulting in the lowest error to the one leading to the highest one. We assign one point to the first speed, two to the second and so on. For each metric, we sum the points of the same speed for the four different providers. Table 8.2 provides a visual representation of the results.

For each metric, we consider the two speeds that received the lowest number of points (highlighted in blues in Table 8.2). This results in the set 60,70,80,100, 120,130. For these speeds, we test all the possible combinations among the elements. We run RTT*locator* using each combination for each RESIP. We apply the same point system described above to find the combination that minimizes median, standard deviation and IQR. At the end of this process, the *Set of Speeds* {60,70,120,130} results in the best result. However, applying this *Set of Speeds* to the files of each RESIP we obtain errors that do not enable accurate geolocation. Table 8.3 reports the median, standard deviation and IQR errors obtained for each RESIP provider. These results clearly show us that this system does not provide the expected results.

While the challenge of determining the optimal *Set of Speeds* for all considered RESIP connections collectively remains unsolved, we opted to employ this *Set of Speeds* on real-world connections to gain initial insights into the achievable outcomes. We executed RTT*locator* with the *Set of Speeds* {60,70,120,130} and kept the other parameters consistent with our previous selections. This process was carried out for each of the four datasets outlined in Section 8.4.1.

Within these datasets, the actual location of the original client is unknown. However, we know that *Airline* X *Dataset* and *Airline* Y *Dataset* were obtained from the same bot signature. The outcomes of RTT*locator* indicate the successful localization of the corresponding clients in North Africa (Morocco and Algeria), hinting at the likelihood of the original client's presence in that vicinity. On the other hand, when we use *Airline* Z *Dataset* as input, we obtain a client location in the middle of the Atlantic Ocean. This clearly shows that this *Set of Speeds* does not obtain a correct result for this set of requests. We have tried to find manually a possible *Set of Speeds* that could give a plausible location for these requests. We discovered that {60, 80, 100, 120, 130} could potentially yield a plausible client location for the *Airline* Z *Dataset*, specifically pinpointing Portugal.

Finally, analyzing *Airline* W *Dataset* we understood that the solution delivered by RTT*locator* was the result of an error. In this dataset, all utilized GATEWAYs are in close proximity to one another. In particular, they fall in the same square in which we divided the planisphere. Consequently, the resultant circles are almost concentric, and the derived location arises not from the intersection of multiple circles, but from their overlapping. Figure 8.9 provides the result of the GBL phase and enable to visualize this problem. Given that a correct multilateration necessitates the intersection of at least three non-coincident circles, this particular dataset fails to meet the prerequisites for such a computation.

Moreover, this opens the door for a new question regarding the problem we are trying to solve: what is the minimum distance that we have to consider as a requirement among at least three GATEWAYs to obtain a plausible result? Furthermore, if the vast majority of GATEWAYs are located in the same area, even if there are at least three different GATEWAY locations, the multiple concentric circles will vastly bias the result. How can we address this problem? In Section 8.5 we propose future approaches to address this issue.

Minimum Circles

Minimum Circles represents the minimum proportion of circles that GBL must successfully extract to determine the completion of this phase. In cases where the count of identified circles contributing to a solution falls below this threshold, it implies that there's a possibility another client might have sent a portion of the incoming requests.

Defining the optimal value of *Minimum Circles* is a task strictly associated with finding the optimal *Set of Speeds*. While we want *Set of Speeds* to reduce the noise in finding a first client location, we do not want it to eliminate those connections related to a second client. As discussed in the previous section, the optimal *Set of Speeds* to find one single solution has not been determined yet, since the candidate one obtained with the points system does not enable us to reach an accurate geolocation. For this reason, we cannot determine the optimal value of this parameter.

However, we have anectodically experimented using the *Set of Speeds* giving us the best results for specific files and merging the files best performing with the same *Set of Speeds* to obtain a set of connections from multiple clients. Our early results from these runs suggest that when two clients are behind a set of requests, *Minimum Circles* set to 50%-60% correctly returns two solutions, in the case of three clients *Minimum Circles* need to be set to 70%-80%. However, when only one client is behind the requests, having a *Minimum Circles* higher than 40%-50% incorrectly results in finding a second location. A possible solution to this problem could be to output for each identified solution the percentage of circles contributing to it. The lower this percentage, the lower the confidence that that location corresponds to a client sending requests and the higher the probability that only the previously identified solutions are the correct ones.

8.5 Discussion and Limitations

Within this chapter, we introduce a method aimed at geolocating a client who transmits requests through a RESIP. To achieve this objective, we leverage a planisphere, a 2D representation of the world. It is important to acknowledge that this creates an inherent limitation. The world is in three dimensions and, consequently, the precise distances between various locations on the globe might undergo minor adjustments when projected onto a 2D surface. Therefore, we need to consider the outcomes produced by our algorithm as an estimation of the actual client location in the 3D space.

Another limitation of our approach is the retrieval of the latitude and longitude of each GATEWAY from a geolocation database (MaxMinf Geo2Lite [70]). If this information was not accurate or incorrect, it could lead *RTTlocator* to an erroneous result. As discussed in Section 2.5 and Section 4.2.4, the accuracy of this database has been questioned in the past [72], [73] but it is recognized that its data is reliable at the country level [74]–[76]. This information tells us that if there are cases in which the GATEWAY location is inaccurate and the area of the corresponding country is small, the error is contained. Moreover, we expect to have few rare large errors in the geolocation of the GATEWAY. In both cases, errors in the client location would result in circles that do not contribute to finding the final location of the client and would simply increase the intrinsic noise created by the problem in hand (Section 8.2). Thus, while acknowledging this limitation, we believe its impact on the technique is negligible.

RTT*locator* geolocation is based on the δ_{RTT}. However, this value could be strategically modified by the original client to avoid the identification of his location. The client would need to delay TLS packets to increase the registered δ_{RTT}. However, this action would have a drawback for the client. It would introduce a delay in obtaining the requested data and this would result in a loss in efficiency on his side.

Our work aims at finding if multiple clients are behind a set of requests. However, if two distinct clients send requests at the same time from the same location in the world, our algorithm recognizes them as one only contribution. Once a strategy to find the optimal *Set of Speeds* will be identified, we will produce a sensitivity analysis to understand what is the minimum distance among clients to enable the geolocation of both contributions.

As discussed in Section 4.4, we know that Mobile TCP Terminating Proxies (MTTPS) break the TCP session between a mobile device and the used antenna. Thus, in those cases where a RESIP leverages a mobile device whose Internet Service Provider uses a MTTP, the obtained δ_{RTT} gives information about the distance between the client and the antenna instead of between the client and the device itself. However, in general, a device is close to the used antenna. Hence, this should not create a large error in our estimation.

As seen in Section 8.4.2, RTT*locator* results can be polluted when all the used GATEWAYs are located close in space. The obtained circles are almost concentric and partially overlap with each other creating wrong solutions. While it is easy to check if the GATEWAY locations are exactly the same as we do in the preliminary phase of GBL (Section 8.3), it is more difficult to understand when the distance between *enough* GATEWAYs is *sufficient* to perform correct multilateration. Moreover, if a large part of GATEWAYs is closed in space and only a few others are in other locations, even if *enough* GATEWAYs are sufficiently far from each other, the result could be biased by the high number of GATEWAYs in one location and we could obtain an erroneous solution. At the same time, we cannot simply discard the contributions of these concentric circles since they give us information about the real solution.

To address this situation, we could model the contributions of these circles considering just one GATEWAY for a defined area in which there is a high concentration of GATEWAYs. We use only the circles corresponding to that GATEWAY but repeated for the number of GATEWAYs in the area. In this way, the crossed squares would have a high count, thus a higher weight in contributing to the solution, but we would limit the partial overlapping of concentric circles. A drawback arises if, among the GATEWAY in this area, some contribute to one solution and some to another. In that case, a second contribution would be lost. Moreover, we need to understand the right strategy to choose which GATEWAY best represents the contribution of the area.

Finally, we saw that the results of the proposed point system do not enable us to have a globally accurate solution Section 8.4.2. We anectodically tried a complementary approach. Instead of running just once *RTTlocator* with the found optimal *Set of Speeds*, we run it five times, each time using a different *Set of Speeds* among the best five found by the point system. We identify the solution as the area of the world in which the majority of the five runs find the solution. Early results on a subset of the ground truth dataset show that the majority of solutions correctly identifies the right area of the world. Thus, we could consider this as an alternative way to find a correct solution. More analyses need to be performed to understand if the approach is consistent and how many runs with different *Set of Speeds* are needed to geolocate the correct solution.

8.6 Summary

In this chapter, we studied the challenging problem of geolocating the client(s) that proxy requests through RESIPs. We propose *RTTlocator*, a multilateration approach modeling the distances between GATEWAYs and the original clients through the δ_{RTT} value of the requests and multiple packet speeds. Our results on both ground truth and real-world scraping datasets suggest that when the correct *Set of Speeds* for a set of requests is identified, *RTTlocator* correctly locates the area of the world where the requests originate from. However, finding a *Set of Speeds* or multiple *Sets of Speeds* that generalize the geolocation is left for further work.

Table 8.2: Points assigned to each tested speed. In blue, we find the lowest points sums.

Median Error															
	Speed (km/ms)														
RESIP	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**	**130**	**140**	**150**	**160**
Bright Data	11	7	6	5	3	1	13	15	14	12	10	2	4	9	8
Oxylabs	15	14	12	10	11	9	7	6	4	1	2	3	5	8	13
Proxyrack	10	13	12	14	11	15	9	8	7	4	1	2	6	3	5
Smartproxy	13	12	11	9	10	4	1	2	3	5	7	6	8	14	15
Total	49	46	41	38	35	29	30	31	28	22	20	13	23	34	41
Standard Deviation Error															
	Speed (km/ms)														
RESIP	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**	**130**	**140**	**150**	**160**
Bright Data	11	14	15	12	10	13	9	8	7	6	5	1	3	2	4
Oxylabs	9	8	7	1	2	5	3	4	6	10	11	13	12	14	15
Proxyrack	11	15	10	9	6	4	8	3	1	5	2	7	14	12	13
Smartproxy	4	5	1	2	3	9	6	10	7	11	12	8	13	14	15
Total	35	42	33	24	21	31	26	25	21	32	30	29	42	42	47
Interquartile Range Error															
	Speed (km/ms)														
RESIP	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**	**130**	**140**	**150**	**160**
Bright Data	13	14	15	12	6	11	9	10	5	8	1	3	7	4	2
Oxylabs	9	7	4	6	10	1	2	3	5	8	12	13	11	14	15
Proxyrack	12	7	6	4	9	2	3	5	1	11	10	13	15	8	14
Smartproxy	1	4	7	3	2	5	6	8	10	9	13	11	12	14	15
Total	35	32	32	25	27	19	20	26	21	36	36	40	45	40	46

Table 8.3: Results of RTT*locator* with the *Set of Speeds* {60,70,120,130}.

RESIP	Median Error (km)	Standard Deviation Error (km)	IQR Error (km)
Bright Data	3084.47	5141.13	5823.17
Oxylabs	3647.14	1554.21	2743.61
Proxyrack	7060.38	3600.65	2814.90
Smartproxy	4837.14	3334.61	2402.79

Chapter 9

Future Works and Conclusion

In this thesis, we introduced innovative approaches to detect and mitigate scraping bots, as well as novel characterizations of the latest technologies used by scrapers and a novel method aimed at determining the geographical origins of proxied scraping requests.

These new approaches give us an advantage in the ongoing war against scrapers. However, the ever-changing and dynamic essence of scraping bots presents challenges in definitively overcoming them. This demands a continuous need to enhance our defense approaches and adjust to their ever-evolving tactics. In the following, we highlight the contributions of this thesis and how we can build upon them to move forward toward fighting scrapers.

Chapter 3 described the idea behind our two innovative server-side detection techniques for requests proxied through RESIP networks, RTT_DETECTION and MRP_DETECTION. Differing from the only other RESIP detection technique described in the state-of-the-art, our methods provide a systematic and deterministic way to identify a RESIP connection by examining only a single request. Furthermore, MRP_DETECTION has the capability to recognize connections from specific sets of RESIP providers. The ideas underlying both techniques are protected in two filed patents and RTT_DETECTION has been implemented in one of the most used anti-bot solutions on the market.

Chapters 4 and 5 detailed the experiment we performed to prove the validity and feasibility of the, respectively, RTT_DETECTION and MRP_DETECTION techniques. In Chapter 4, we consistently observed that the δ_{RTT} (calculated as RTT$_{TLS}$- RTT$_{TCP}$) exhibited significantly higher values for RESIP connections compared to direct connections when the same client was utilized in both scenarios. Initial results indicated 50ms as the most effective threshold in differentiating the two types of connections.

Our analyses of real-world scraping connections and specific client environments identified the possibility of false positives. However, our results suggest that increasing the threshold for mobile network connections could effectively reduce the number of false positives caused by Mobile TCP Terminating Proxies.

In Section 4.3.6, we saw that different providers have a different distribution of the δ_{RTT}. Building upon this, we could explore the potential of utilizing the δ_{RTT} not only for detecting whether a connection has traversed a RESIP network, but also for discerning the specific RESIP provider through which the request has been proxied. Another future work regarding RTT_DETECTION, associated to this analysis, implies the expansions of the client environment analyses performed in Section 4.4.1 and the differentiation of RESIP and false positive connections based on the different δ_{RTT} averages (as we suggest for Mobile TCP Terminating Proxies) or complementary information (MTU analysis to identify VPNs).

Chapter 5 described the experiment to validate MRP_DETECTION. The technique leverages a specific machine retransmission protocol used by two of the analyzed providers allegedly to increase efficiency. Moreover, this chapter revealed the internal algorithm all RESIP providers use when the server does not acknowledge packets. They involve 3-4 GATEWAYs from different areas of the world to try to perform the connection. This suggests that they try to circumvent possible regional problems and restrictions.

Most likely, the specific machine retransmission protocol used by the two RESIP providers is not the only type of operation in which RESIP providers try to optimize their actions. A possible future work consists of testing connections from different RESIP providers in atypical situations, e.g. producing long delays in the middle of a connection or requiring the client to open a new connection from a different port, to check if specific behaviors take place at the network layer and build detection/attribution methods based on them.

In Chapter 6, we proposed a new deceptive approach based on the redirection of scraping requests to a honeypot mimicking the real website. Thanks to this, we were able to serve modified information to scrapers for almost two months.

A natural follow-up of this work is a larger usage of the honeypot system to expand the analyses performed so far and test new data poisoning strategies for scrapers. The positive results of our experiment convinced our partner, Amadeus IT Group, to take action on this proposition. Work is underway for a honeypot environment directly built into their production system.

In Chapter 7, we characterized the RESIP ecosystem. We presented various facets of the RESIP infrastructures, emphasizing both the shared characteristics and distinctions among providers in terms of geographical distribution and types, management and amount of machines. We highlighted the significant similarities among two of the considered providers in all the different RESIP studies we performed.

Moreover, we proposed a novel idea for a potential IP-based RESIP detection methodology at the /24 level. However, this approach might not be effective since there is no general rule among ASes to maintain the previous prefix after reallocation. Future works in this area would involve conducting real-world measurements to evaluate the effectiveness and potential constraints of the proposed methodology.

Additionally, in the same chapter, we estimated the size of residential IPs available to RESIP providers. Our results did not corroborate the claims RESIP providers do on their respective websites.

A follow-up of this work would be to understand the actual number of devices corresponding to these IP addresses and their evolution in time. Expanding on our current model and a recent work about estimating the number of devices behind the IP addresses used in botnet attacks [140], we could better understand the size of the pool of devices available to RESIPs.

Chapter 8 delved into the intricate challenge of determining the geographical location of the client(s) that proxy requests through RESIPs. It introduced *RTTlocator*, a multilateration technique that models the distances between GATEWAYs and the original clients by utilizing the δ_{RTT} value of the requests along with multiple packet speeds. Our findings, derived from both ground truth and real-world scraping datasets, indicate that once the correct *Set of Speeds* is identified for a given set of requests, *RTTlocator* accurately pinpoints the global region of origin. However, the task of identifying a *Set of Speeds* or multiple *Sets of Speeds* that can generalize the geolocation process remains a subject for future research. Correctly defining the parameters that enable us to accurately geolocate different sets of requests is a natural future work idea related to this contribution.

Finally, the contributions of this thesis focus on the specific use cases of defeating scraping bots. However, they can be easily applied to other categories of bots and attacks to identify and mitigate them. We can leverage the deceptive mitigation solution we presented in Chapter 6 to lure attackers into believing they are accessing genuine content, while they are obtaining fake data. One context in which we can take advantage of this is account takeover [145]. In this type of attack, brute force is generally used to identify the right combination of username and password to perform a login. Once we identify this attack is happening, we can lure the attacker into believing that he has found the right combination of credentials. We can then send him to a honeypot where we can study their actions, deny the actual attack to happen and provide them with poisoned information, as previously done, in a different context, by Alata et al. [146]. Furthermore, we can leverage honeytokens to possibly detect where this information is then used on the

Internet.

Additionally, we could use this system to counter denial of inventory attacks, the act of making selections and retaining items from a constrained inventory or stock, without finalizing any purchase [7]. This attack is usually performed using a large number of bots and causes the items to become virtually out-of-stock. This creates economic damage and prevents genuine users from purchasing the items. When this type of attack is detected, we could redirect these requests to a honeypot mimicking the real system. In this way, the attacker believes they have reached their goal, while the items remain available for genuine users.

On top of the above-mentioned contributions in detecting and mitigating web scraping activity, we believe that one of the merits of this thesis is to bring light to internal characteristics of Residential IP Proxies and find new ways to detect server-side when scrapers, and attackers in general, use them. Because of their favorable characteristics for attackers (no need to build/maintain a private botnet and usage of IPs with a good reputation), we believe these parties will be more and more leveraged for malicious activities in the near future. Thus, our findings can be used for protecting websites not only from scraping attacks but from a more diverse set of malicious actions. We hope that the insights and contributions of this thesis can serve as a starting point for future work in the field, as well as in web scraping detection and mitigation, and assist academics and practitioners to better deal with these threats.

Author's Publications

Conferences

[C1] **E. Chiapponi**, M. Dacier, O. Thonnard, M. Fangar, M. Mattsson, and V. Rigal, "An industrial perspective on web scraping characteristics and open issues," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2022, pp. 5–8.

[C2] **E. Chiapponi**, M. Dacier, O. Thonnard, M. Fangar, and V. Rigal, "BADPASS: Bots Taking ADvantage of Proxy as a Service," in *Information Security Practice and Experience*, C. Su, D. Gritzalis, and V. Piuri, Eds., Cham: Springer International Publishing, pp. 327–344, ISBN: 978-3-031-21280-2.

[C3] **E. Chiapponi**., M. Dacier, and O. Thonnard, "Towards detecting and geolocalizing web scrapers with round trip time measurements," in *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*, 2023, pp. 1–4.

[C4] **E. Chiapponi**, M. Dacier, and O. Thonnard, "Poster: The impact of the client environment on residential ip proxies detection," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, ser. IMC '23, Montreal QC, Canada: Association for Computing Machinery, 2023, pp. 712–713.

[C5] **E. Chiapponi**., M. Dacier, and O. Thonnard, "Inside Residential IP Proxies: Lessons Learned from Large Measurement Campaigns," in *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2023, pp. 501–512.

[C6] **E. Chiapponi**, O. Catakoglu, O. Thonnard, and M. Dacier, "HoPLA: A Honeypot Platform to Lure Attackers," in *CAESAR 2020, Computer & Electronics Security Applications Rendez-vous, Deceptive security Conference, part of European Cyber Week*, 2020.

[C7] **E. Chiapponi**, M. Dacier, M. Todisco, O. Catakoglu, and O. Thonnard, "Botnet sizes: When maths meet myths," in *Service-Oriented Computing – ICSOC 2020 Workshops*, H. Hacid, F. Outay, H.-y. Paik, *et al.*, Eds., Cham: Springer International Publishing, 2020, pp. 596–611, ISBN: 978-3-030-76352-7.

[C8] M. Champion, M. Dacier, and **E. Chiapponi**, "ImMuNE: Improved Multilateration in Noisy Environments," in *2022 IEEE 11th International Conference on Cloud Networking (CloudNet)*, 2022, pp. 1–6.

Journals

[J1] **E. Chiapponi**, M. Dacier, O. Catakoglu, O. Thonnard, and M. Todisco, "Scraping Airlines Bots: Insights Obtained Studying Honeypot Data," *International Journal of Cyber Forensics and Advanced Threat Investigations*, vol. 2, no. 1, pp. 3–28, 2021, ISSN: 2753-9997.

Submitted Patents

[P1] **E. Chiapponi**, M. Dacier, O. Thonnard, V. Rigal, and M. Fangar, "PROXY DETECTION SYSTEMS AND METHODS," United States Patent Application No. 17/746556, Applicant(s): AMADEUS S.A.S. 2022.

[P2] **E. Chiapponi**, M. Dacier, O. Thonnard, V. Rigal, and M. Fangar, "PROXY DETECTION SYSTEMS AND METHODS," United States Patent Application No. 63/497053, Applicant(s): AMADEUS S.A.S. 2023.

References

[1] DataDome. "Advanced techniques for detecting & blocking sneaker bots." (2023), [Online]. Available: https://datadome.co/bot-management-protection/how-to-detect-block-manage-sneaker-bots/.

[2] S. Dietrich, N. Long, and D. Dittrich, "Analyzing distributed denial of service tools: The shaft case.," in *Proceedings of the 14th USENIX conference on System administration*, New Orleans, Louisiana, USA, 2000, pp. 329–339.

[3] M. Antonakakis, T. April, M. Bailey, *et al.*, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110, ISBN: 978-1-931971-40-9.

[4] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, ser. LEET'08, San Francisco, California: USENIX Association, 2008.

[5] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in ddos attacks: Trends and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2242–2270, 2015.

[6] L. Zhang, S. Yu, D. Wu, and P. Watters, "A survey on latest botnet attack and defense," in *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 2011, pp. 53–60.

[7] C. Watson and T. Zaw, "OWASP Automated Threat Handbook Web Applications," OWASP Foundation, Tech. Rep., 2018.

[8] Wikipedia. "Hiq labs v. linkedin." (2023), [Online]. Available: https://en.wikipedia.org/wiki/HiQ_Labs_v._LinkedIn.

[9] Imperva, “Bad Bot Report 2021, The Pandemic of the Internet,” Imperva, Tech. Rep., 2021.

[10] Imperva, “Imperva Bad Bot Report,” Imperva, Tech. Rep., 2020.

[11] B. Gelbord. “In China and Japan, Malicious Botnets Surge Amidst Holiday Ecommerce Traffic.” (2022), [Online]. Available: `https://www.akamai.com/blog/security/china-and-japan-holiday-botnets`.

[12] DataDome. “Cost of Bots Calculator.” (2023), [Online]. Available: `https://datadome.co/cost-of-bot-calculator/`.

[13] “Scrapy.” (2023), [Online]. Available: `https://scrapy.org/`.

[14] “Phantom js.” (2023), [Online]. Available: `https://phantomjs.org/`.

[15] “Selenium.” (2023), [Online]. Available: `https://www.selenium.dev/`.

[16] R. Egger, M. Kroner, and A. Stöckl, “Web scraping,” in *Applied Data Science in Tourism: Interdisciplinary Approaches, Methodologies, and Applications*, R. Egger, Ed. Cham: Springer International Publishing, 2022, pp. 67–82, ISBN: 978-3-030-88389-8.

[17] B. Amin Azad, O. Starov, P. Laperdrix, and N. Nikiforakis, “Web runner 2049: Evaluating third-party anti-bot services,” in *Proc. of DIMVA 2020*.

[18] H. Debar, M. Dacier, and A. Wespi, “A revised taxonomy for intrusion-detection systems,” in *Annales des télécommunications*, Springer, vol. 55, 2000, pp. 361–378.

[19] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, “Browser fingerprinting: A survey,” *ACM Trans. Web*, vol. 14, no. 2, 2020, ISSN: 1559-1131.

[20] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna, “PUBCRAWL: Protecting users and businesses from CRAWLers,” in *21st USENIX Security Symposium (USENIX Security 12)*, Bellevue, WA: USENIX Association, Aug. 2012, pp. 507–522.

[21] A. G. Lourenço and O. O. Belo, “Catching web crawlers in the act,” in *Proceedings of the 6th International Conference on Web Engineering*, ser. ICWE ’06, Palo Alto, California, USA: Association for Computing Machinery, 2006, pp. 265–272, ISBN: 1595933522.

[22] A. Vastel, W. Rudametkin, R. Rouvoy, and X. Blanc, “FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers,” in *Proc. of MADWeb’20*, San Diego, United States.

[23] E. Bursztein, A. Malyshev, T. Pietraszek, and K. Thomas, “Picasso: Lightweight device class fingerprinting for web clients,” in *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM ’16, Vienna, Austria: Association for Computing Machinery, 2016, pp. 93–102, ISBN: 9781450345644.

[24] H. Jonker, B. Krumnow, and G. Vlot, “Fingerprint surface-based detection of web bot detectors,” in *Computer Security – ESORICS 2019*, K. Sako, S. Schneider, and P. Y. A. Ryan, Eds., Cham: Springer International Publishing, 2019, pp. 586–605, ISBN: 978-3-030-29962-0.

[25] X. Li, B. A. Azad, A. Rahmati, and N. Nikiforakis, “Good bot, bad bot: Characterizing automated browsing activity,” in *2021 IEEE Symposium on Security and Privacy (S&P)*, 2021, pp. 1589–1605.

[26] L. Brotherston. “Tls fingerprinting.” (2016), [Online]. Available: `https://github.com/LeeBrotherston/tls-fingerprinting`.

[27] Z. Durumeric, Z. Ma, D. Springall, *et al.*, “The Security Impact of HTTPS Interception,” in *24th Network and Distributed System Security Symposium (NDSS)*, 2017.

[28] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, “CAPTCHA: Using hard AI problems for security,” in *Advances in Cryptology EUROCRYPT 2003*, E. Biham, Ed., Springer, 2003, pp. 294–311.

[29] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, “Re: Captchas: Understanding captcha-solving services in an economic context,” in *Proc. USENIX Security 2010*, USA, p. 28.

[30] M. Egele, L. Bilge, E. Kirda, and C. Kruegel, “Captcha smuggling: Hijacking web browsing sessions to create captcha farms,” in *Proc. of the 2010 ACM SAC conf.*, Sierre, Switzerland, 2010, pp. 1865–1870, ISBN: 9781605586397.

[31] 2Captcha, “Captcha typing job. online earning by solving captchas,” 2023.

[32] N. Singh. “How to Fight Bad Bots and Win: Radware’s New Crypto Mitigation Algorithms.” (2022), [Online]. Available: `https://www.radware.com/blog/application-protection/2022/07/how-to-fight-bad-bots-and-win-radwares-new-crypto-mitigation-algorithms/`.

[33] Imperva. “Advanced bot protection general release notes.” (2023), [Online]. Available: `https://docs.imperva.com/bundle/advanced-bot-protection/page/76994.htm`.

[34] "Amadeus Global Report 2020," Amadeus IT Group, Tech. Rep., 2021.

[35] Imperva, "2022 Bad Bot Report, Evasive Bots Drive Online Fraud," Imperva, Tech. Rep., 2022.

[36] DataDome. "Blocking the IP is Not Enough - How to Stop Bots on Residential IPs." (2022), [Online]. Available: `https://datadome.co/bot-management-protection/one-third-bad-bots-using-residential-ip-addresses/`.

[37] Bright Data. "Web Unlocker Avoid Getting Blocked." (2023), [Online]. Available: `https://brightdata.com/products/web-unlocker`.

[38] X. Mi, X. Feng, X. Liao, *et al.*, "Resident evil: Understanding residential IP proxy as a dark service," in *2019 IEEE Symposium on Security and Privacy (S&P)*, 2019, pp. 1185–1201.

[39] X. Mi, S. Tang, Z. Li, X. Liao, F. Qian, and X. Wang, "Your Phone is My Proxy: Detecting and Understanding Mobile Proxy Networks," en, in *Proc. of NDSS 2021*, 2021.

[40] N. Abdurahiman, "Towards Residential Proxies Detection: An Experimental Analysis in the Android Environment," M.S. thesis, Hamad Bin Khalifa University, 2021.

[41] M. Frappier, P. Plante, and G. Joly. "Illegitimate residential proxy services: the case of 911.re and its IOCs." (2022), [Online]. Available: `https://gric.recherche.usherbrooke.ca/rpaas/`.

[42] A. Vastel. "Ever wonder how proxy providers & BaaS providers obtain residential proxies?" (2022), [Online]. Available: `https://datadome.co/bot-detection/how-proxy-providers-get-residential-proxies`.

[43] A. Tosun, M. De Donno, N. Dragoni, and X. Fafoutis, "RESIP Host Detection: Identification of Malicious Residential IP Proxy Flows," in *2021 IEEE International Conference on Consumer Electronics (ICCE)*, 2021, pp. 1–6.

[44] B. Krebs. "The Rise of "Bulletproof" Residential Networks." (2019), [Online]. Available: `https://krebsonsecurity.com/2019/08/the-rise-of-bulletproof-residential-networks/`.

[45] M. Yang, Y. Yu, X. Mi, *et al.*, "An Extensive Study of Residential Proxies in China," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[46] W. Turton. "Russian hackers used home networks to evade detection." (2021), [Online]. Available: `https://www.bloomberg.com/news/articles/2021-10-26/suspected-russian-hackers-use-home-networks-to-evade-detection`.

[47] E. Khan, A. Sperotto, J. van der Ham, and R. van Rijswijk-Deij, "Stranger VPNs: Investigating the Geo-Unblocking Capabilities of Commercial VPN Providers," in *Passive and Active Measurement*, A. Brunstrom, M. Flores, and M. Fiore, Eds., Cham: Springer Nature Switzerland, 2023, pp. 46–68, ISBN: 978-3-031-28486-1.

[48] J. Choi, M. Abuhamad, A. Abusnaina, *et al.*, "Understanding the proxy ecosystem: A comparative analysis of residential and open proxies on the internet," *IEEE Access*, vol. 8, pp. 111 368–111 380, 2020.

[49] A. Hanzawa and H. Kikuchi, "Analysis on Malicious Residential Hosts Activities Exploited by Residential IP Proxy Services," in *Information Security Applications*, Springer International Publishing, 2020, pp. 349–361.

[50] A. Vastel. "How to Use Machine Learning to Detect Residential Proxies." (2022), [Online]. Available: `https://datadome.co/bot-management-protection/how-to-use-machine-learning-to-detect-residential-proxies/`.

[51] H. Hoogstraaten, "Evaluating server-side internet proxy detection methods," M.S. thesis, Leiden University, 2018.

[52] N. Tschacher. "Is this a valid method to detect Proxies?" (2021), [Online]. Available: `https://incolumitas.com/2021/11/26/is-this-a-valid-method-to-detect-proxies/`.

[53] A. T. Webb and A. L. Narasima Reddy, "Finding proxy users at the service using anomaly detection," in *IEEE Conference on Communications and Network Security (CNS)*, 2016, pp. 82–90.

[54] A. Turgeman, Y. Lehmann, Y. Azizi, and I. Novick, "Detection of proxy server," Patent, United States, US10069837B2, 2019.

[55] B. Cheswick, "An Evening with Berferd in which a cracker is Lured, Endured, and Studied," in *Proc. Winter USENIX Conference*, San Francisco, CA, USA, 1992, pp. 20–24.

[56] W. Z. Venema, "TCP Wrapper: Network Monitoring, Access Control, and Booby Traps.," in *USENIX Summer*, 1992.

[57] F. Cohen, "The use of deception techniques: Honeypots and decoys," *Handbook of Information Security*, vol. 3, no. 1, pp. 646–655, 2006.

[58] C. Leita and M. Dacier, “SGNET: a worldwide deployable framework to support the analysis of malware threat models,” in *2008 Seventh European Dependable Computing Conference*, IEEE, 2008, pp. 99–109.

[59] F. Pouget, M. Dacier, and H. Debar, “White paper: honeypot, honeynet, honeytoken: terminological issues,” EURECOM, Tech. Rep. EURECOM+1275, 2003.

[60] F. Pouget and M. Dacier, “Honeypot-based forensics,” in *AusCERT Asia Pacific Information Technology Security Conference*, 2004.

[61] V. Nicomette, M. Kaaniche, E. Alata, and M. Herrb, “Set-up and deployment of a high-interaction honeypot: Experiment and lessons learned,” 2, vol. 7, Springer, 2011, pp. 143–157.

[62] O. Thonnard and M. Dacier, “Actionable knowledge discovery for threats intelligence support using a multi-dimensional data mining methodology,” in *2008 IEEE International Conference on Data Mining Workshops*, IEEE, 2008, pp. 154–163.

[63] O. Thonnard, W. Mees, and M. Dacier, “Addressing the attack attribution problem using knowledge discovery and multi-criteria fuzzy decision-making,” in *Proceedings of the ACM SIGKDD workshop on CyberSecurity and intelligence informatics*, 2009, pp. 11–21.

[64] “LaBrea: "Sticky" Honeypot and IDS.” (), [Online]. Available: `https://labrea.sourceforge.io/labrea-info.html`.

[65] M. Delong, E. Filiol, and B. David, “Investigation and surveillance on the darknet: An architecture to reconcile legal aspects with technology,” in *ECCWS 2019 18th European Conference on Cyber Warfare and Security*, Academic Conferences and publishing limited, 2019, p. 151.

[66] “The Bait and Switch Honeypot.” (2020), [Online]. Available: `%7Bhttp://baitnswitch.sourceforge.net/%7D`.

[67] K. Takemori, K. Rikitake, Y. Miyake, and K. Nakao, “Intrusion trap system: An efficient platform for gathering intrusion-related information,” in *10th International Conference on Telecommunications, 2003. ICT 2003.*, vol. 1, 2003, 614–619 vol.1.

[68] Collins Dictionaries. “Geolocation.” (2023), [Online]. Available: `https://www.collinsdictionary.com/dictionary/english/geolocation`.

[69] RIPE. “Regional Internet Registry for Europe.” (), [Online]. Available: `www.ripe.net`.

[70] MaxMind. “Geolite2 free geolocation data.” (1999), [Online]. Available: `https://dev.maxmind.com/geoip/geolite2-free-geolocation-data`.

[71] P. Gill, Y. Ganjali, B. Wong, and D. Lie, “Dude, Where’s That IP? Circumventing Measurement-Based IP Geolocation,” in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security’10, Washington, DC: USENIX Association, 2010, p. 16.

[72] P. Callejo, M. Gramaglia, R. Cuevas, and A. Cuevas, “A deep dive into the accuracy of IP Geolocation Databases and its impact on online advertising,” *IEEE Transactions on Mobile Computing*, pp. 1–1, 2022.

[73] M. Gharaibeh, A. Shah, B. Huffaker, H. Zhang, R. Ensafi, and C. Papadopoulos, “A Look at Router Geolocation in Public and Commercial Databases,” in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC ’17, London, United Kingdom: Association for Computing Machinery, 2017, pp. 463–469, ISBN: 9781450351188.

[74] M. Cozar, D. Rodriguez, J. M. Del Alamo, and D. Guaman, “Reliability of IP Geolocation Services for Assessing the Compliance of International Data Transfers,” in *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2022, pp. 181–185.

[75] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye, “IP Geolocation Databases: Unreliable?” *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 53–56, 2011, ISSN: 0146-4833.

[76] M. Schopman, “Validating the accuracy of the MaxMind GeoLite2City database,” Radboud University, 2021.

[77] A. F. V. F. P. Millerand and J. McBrewster, *Multilateration.* 2010, ISBN: 978-6130772307.

[78] T. Wang, K. Xu, J. Song, and M. Song, “An optimization method for the geolocation databases of internet hosts based on machine learning,” *Mathematical Problems in Engineering*, vol. 2015, 2015.

[79] D. Li, J. Chen, C. Guo, *et al.*, “IP-geolocation mapping for involving moderately-connected Internet regions,” *Project participation from Microsoft Research*, 2009.

[80] B. Gueye, S. Uhlig, A. Ziviani, and S. Fdida, “Leveraging buffering delay estimation for geolocation of internet hosts,” in *International Conference on Research in Networking*, Springer, 2006, pp. 319–330.

[81] B. Wong, I. Stoyanov, and E. G. Sirer, "Octant: A comprehensive framework for the geolocalization of internet hosts.," in *NSDI*, vol. 7, 2007, pp. 23–23.

[82] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, "Towards IP geolocation using delay and topology measurements," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006, pp. 71–84.

[83] M. J. Arif, S. Karunasekera, S. Kulkarni, A. Gunatilaka, and B. Ristic, "Internet host geolocation using maximum likelihood estimation technique," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, IEEE, 2010, pp. 422–429.

[84] F. Zhao, Y. Song, F. Liu, K. Ke, J. Chen, and X. Luo, "City-level geolocation based on routing feature," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, IEEE, 2015, pp. 414–419.

[85] M. Grey, D. Schatz, M. Rossberg, and G. Schaefer, "Towards distributed geolocation by employing a delay-based optimization scheme," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2014, pp. 1–7.

[86] S. Laki, P. Mátray, P. Hága, T. Sebők, I. Csabai, and G. Vattay, "Spotter: A model based active geolocation service," in *2011 Proceedings IEEE INFOCOM*, IEEE, 2011, pp. 3173–3181.

[87] Z. Weinberg, S. Cho, N. Christin, V. Sekar, and P. Gill, "How to catch when proxies lie: Verifying the physical locations of network proxies with active geolocation," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 203–217.

[88] Z. Dong, R. D. Perera, R. Chandramouli, and K. Subbalakshmi, "Network measurement based modeling and optimization for IP geolocation," *Computer Networks*, vol. 56, no. 1, pp. 85–98, 2012.

[89] T. Chung, D. Choffnes, and A. Mislove, "Tunneling for Transparency: A Large-Scale Analysis of End-to-End Violations in the Internet," in *Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 199–213.

[90] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, *et al.*, "A longitudinal, End-to-End view of the DNSSEC ecosystem," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1307–1322.

[91] Bright Data. "Bright Data - The World's #1 Web Data Platform." (2023), [Online]. Available: `https://brightdata.com/`.

[92] Oxylabs. “Innovative Proxy Service to Gather Data at Scale | Oxylabs.” (2023), [Online]. Available: `https://oxylabs.io/`.

[93] Proxyrack. “Proxyrack: Buy Proxies HTTP, UDP, SOCKS Proxy.” (2023), [Online]. Available: `https://www.proxyrack.com/`.

[94] Smartproxy. “Smartproxy - Best Value Proxies & Data Collection Solutions.” (2023), [Online]. Available: `https://smartproxy.com/`.

[95] R. Zullo, A. Pescapé, K. Edeline, and B. Donnet, “Hic sunt proxies: Unveiling proxy phenomena in mobile networks,” in *2019 Network Traffic Measurement and Analysis Conference (TMA)*, 2019, pp. 227–232.

[96] N. Weaver, C. Kreibich, M. Dam, and V. Paxson, “Here be web proxies,” in *Passive and Active Measurement*, M. Faloutsos and A. Kuzmanovic, Eds., Cham: Springer International Publishing, 2014, pp. 183–192, ISBN: 978-3-319-04918-2.

[97] W. R. Stevens and K. R. Fall, *TCP/IP Illustrated, Volume 1: The Protocols (Second edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2012, ISBN: 978-0201633467.

[98] D. El Kaim - Bright Data, Private Email Communication, 25-01-2022.

[99] AWS. “Amazon Lightsail.” (2023), [Online]. Available: `https://aws.amazon.com//lightsail/`.

[100] Microsoft. “Azure.” (2023), [Online]. Available: `https://azure.microsoft.com/`.

[101] Python Software Foundation. “Python.” (2023), [Online]. Available: `https://www.python.org/`.

[102] Python Software Foundation. “http.server.” (2023), [Online]. Available: `https://github.com/python/cpython/blob/3.10/Lib/http/server.py/`.

[103] Python Software Foundation. “urllib.” (2023), [Online]. Available: `https://github.com/python/cpython/tree/3.10/Lib/urllib/`.

[104] KiwiNet. “Pyshark.” (2023), [Online]. Available: `https://github.com/KimiNewt/pyshark`.

[105] R. Oppliger, *SSL and TLS: Theory and Practice, Second Edition*, 2nd. USA: Artech House, Inc., 2016, ISBN: 1608079988.

[106] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC Editor, RFC 5246, 2008, `http://www.rfc-editor.org/rfc/rfc5246.txt`.

[107] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC Editor, RFC 8446, 2018.

[108] The PostgreSQL Global Development Group. "PostgreSQL." (1996), [Online]. Available: `https://www.postgresql.org/`.

[109] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive os fingerprinting methods in the jungle of wireless networks," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–9.

[110] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky. "Passiveosfingerprint." (2018), [Online]. Available: `https://github.com/CSIRT-MU/PassiveOSFingerprint`.

[111] Wikipedia. "Haversine." (2023), [Online]. Available: `https://en.wikipedia.org/wiki/Haversine_formula`.

[112] W3Techs. "Usage statistics of Default protocol HTTPS for websites." (2023), [Online]. Available: `https://w3techs.com/technologies/details/ce-httpsdefault`.

[113] Jack Wherry. "What is WireGuard?" (2023), [Online]. Available: `https://cybernews.com/what-is-vpn/wireguard-protocol/#:~:text=WireGuard%5C%20is%5C%20a%5C%20VPN%5C%20protocol,reasons%5C%20why%5C%20it's%5C%20so%5C%20fast..`

[114] The Tor Project. "ExoneraTor." (2023), [Online]. Available: `https://metrics.torproject.org/exonerator.html/`.

[115] SFR. "SFR." (2023), [Online]. Available: `https://www.sfr.fr/`.

[116] NordVPN. "NordVPN." (2023), [Online]. Available: `https://`.

[117] Google. "Google Chrome." (2023), [Online]. Available: `https://www.google.com/intl/en/chrome/`.

[118] Microsoft. "Microsoft Edge." (2023), [Online]. Available: `https://www.microsoft.com/en-us/edge`.

[119] Oberlo. "Most Popular Web Browsers in 2023." (2023), [Online]. Available: `https://www.oberlo.com/statistics/browser-market-share`.

[120] IPQualityScore. "IPQUALITYSCORE Proactively Prevent Fraud." (2023), [Online]. Available: `https://www.ipqualityscore.com/`.

[121] ValdikSS. "Detecting VPN (and its configuration!) and proxy users on the server side." (2015), [Online]. Available: `https://medium.com/@ValdikSS/detecting-vpn-and-its-configuration-and-proxy-users-on-the-server-side-1bcc59742413`.

[122] Statista. “Virtual Private Network (VPN) usage in the United States in 2022.” (2023), [Online]. Available: https://www.statista.com/statistics/1342696/vpn-usage-united-states/.

[123] L. Ceci. “Mobile internet usage worldwide - Statistics & Facts.” (2023), [Online]. Available: https://www.statista.com/topics/779/mobile-internet/#topicOverview.

[124] Digital Element. “Carrier Data Insights.” (2023), [Online]. Available: https://www.digitalelement.com/solutions/user-context/carrier-data/.

[125] DataDome. “8 ways to effectively reduce server response time.” (2022), [Online]. Available: https://datadome.co/learning-center/how-to-reduce-server-response-time/.

[126] E. Chiapponi, “Building a honeypot to mitigate bad bot traffic,” M.S. thesis, Politecnico di Torino - TELECOM Paris, 2020.

[127] Python Software Foundation. “pydnsbl 0.5.4.” (2023), [Online]. Available: https://pypi.org/project/pydnsbl/0.5.4/.

[128] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota, “Birthday Paradox for Multi-collisions,” in *Information Security and Cryptology – ICISC*, Berlin, Heidelberg, 2006, pp. 29–40.

[129] Wikipedia. “Hilbert curve.” (2023), [Online]. Available: https://en.wikipedia.org/wiki/Hilbert%5C_curve.

[130] D. Wessels. “Ipv4-heatmap.” (2011), [Online]. Available: https://github.com/measurement-factory/ipv4-heatmap.

[131] ARIN. “American Registry for Internet Numbers.” (2023), [Online]. Available: https://www.arin.net/.

[132] LACNIC. “Latin America and Caribbean Network Information Centre.” (2023), [Online]. Available: https://www.lacnic.net/.

[133] APNIC. “Asia Pacific Network Information Centre.” (2023), [Online]. Available: https://www.apnic.net/.

[134] AFRINIC. “African Network Information Centre.” (2023), [Online]. Available: https://afrinic.net/.

[135] J. M. Allen, “Os and application fingerprinting techniques,” in *SANS Institute InfoSec Reading Room*, 2007.

[136] O. A. Osanaiye and M. Dlodlo, “TCP/IP header classification for detecting spoofed DDoS attack in Cloud environment,” in *IEEE EUROCON 2015 - International Conference on Computer as a Tool (EUROCON)*, 2015, pp. 1–6.

[137] M. Zalewski. “P0f v3.” (2016), [Online]. Available: https://lcamtuf.coredump.cx/p0f3/.

[138] R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein, “Passive Operating System Identification From TCP / IP Packet Headers,” in *Proceedings Workshop on Data Mining for Computer Security (DM-SEC)*, 2003.

[139] R. Padmanabhan, A. Dhamdhere, E. Aben, k. claffy kc, and N. Spring, “Reasons Dynamic Addresses Change,” in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC ’16, Santa Monica, California, USA: Association for Computing Machinery, 2016, pp. 183–198, ISBN: 9781450345262.

[140] L. Böck, D. Levin, R. Padmanabhan, C. Doerr, and M. Mühlhäuser, “How to Count Bots in Longitudinal Datasets of IP Addresses,” in *Proceedings 2023 Network and Distributed System Security Symposium*, San Diego, CA, USA: Internet Society, 2023, ISBN: 978-1-891562-83-9.

[141] Python. “Scipy.optimize Curve fit function.” (2023), [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html.

[142] S. M. Stigler, “Francis Galton’s Account of the Invention of Correlation,” *Statistical Science*, vol. 4, no. 2, pp. 73–79, 1989, ISSN: 08834237.

[143] H. Griffioen and C. Doerr, “Quantifying Autonomous System IP Churn Using Attack Traffic of Botnets,” in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES ’20, Virtual Event, Ireland: Association for Computing Machinery, 2020, ISBN: 9781450388337.

[144] A. Augustin. “Easy-trilateration.” (2021), [Online]. Available: https://github.com/agusalex/easy-trilateration.

[145] Imperva. “Account Takeover.” (2023), [Online]. Available: https://www.imperva.com/learn/application-security/account-takeover-ato/.

[146] E. Alata, V. Nicomette, M. Kaaniche, M. Dacier, and M. Herrb, “Lessons learned from the deployment of a high-interaction honeypot,” in *2006 Sixth European Dependable Computing Conference*, 2006, pp. 39–46.