Imperial College London

BENG INDIVIDUAL PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Identification of IP addresses using fraudulent geolocation data

Author:
James Williams

Supervisor:
Dr. Sergio Maffeis

Second Marker:
Mr. Dominik Harz

June 15, 2020

Abstract

IP geolocation information is used all over the internet, but is easily faked. A number of different internet organisations do this – from bulletproof hosting providers attempting to conceal the location of their servers, to VPN providers looking to sell services in countries they don't have a presence in. Servers using fraudulent IP geolocation in this way may also be more likely to be hosting fraudulent content, making IP geolocation fraud important to detect in the context of internet fraud prevention. In this project, a system has been developed for detecting this kind of IP geolocation fraud.

The system developed in this report uses measurements from a global network of measurement servers – an array of 8 servers in 7 different countries managed by Netcraft, and over 10,000 servers in 176 countries through the RIPE Atlas API.

Using this system we have analysed the prevalence of geolocation fraud in address space spanning over 4 million IPs, which is, to the best of our knowledge, the largest study of its kind conducted. Despite focusing on only a small part of the IPv4 address space, our analysis has revealed incorrect geolocation being used by over 62,000 internet hosts, targeting 225 out of the 249 possible country codes. In terms of address space, we have discovered incorrect geolocation being used by IP address blocks cumulatively spanning over 2.1 million IPs.

This project has been conducted as an industrial project with Netcraft. The system described in this report has been deployed internally at Netcraft, and will continue to be used after this project's conclusion.

Acknowledgements

I would like to thank Dr. Sergio Maffeis for his supervision and guidance throughout this project. I'd also like to thank Netcraft for their support and expertise – in particular, I'd like to thank Graham Edgecombe, Tom Harwood and Mike Prettejohn, all of whom have been very helpful. Their knowledge in the area of internet security and networking has been invaluable while working on this project.

Finally, I thank my parents, my family and my friends for supporting me during my time at Imperial.

Contents

Chapter 1

Introduction

Location, Location, Location – this famous proverb, dating back to 1926 [1], underlines the importance of location in the property industry. Location information is also increasingly being used on the internet by many parties, from advertisers and online shops to streaming platforms [2, 3, 4]. The relative privacy and anonymity of the internet is perhaps one of its best features, but in some situations being able to determine the location of a host is of great importance:

- **Fraud Detection** – knowing whether a user's location corresponds to their shipping or billing address can help in the detection of fraud [5].
- **Classification of Phishing Websites** – if it can be determined that a host has fake geolocation data, it is more likely that it is being used to host phishing sites [6].
- **Criminal Investigations** – in many cases, nations are unable to find and prosecute cybercriminals due to incorrect geolocation data [7]. The US Federal Trade Commission states that inaccurate geolocation data "has posed significant obstacles in FTC investigations" [8].

There are a number of reasons why network operators might want to use incorrect geolocation for a block of IP addresses:

- **Detection Avoidance** – Phishers and fraudsters would like their actions to not be detected by automated systems, which may use geolocation discrepancies as indicators of foul play.
- **Prosecution Avoidance** – Tor exit node hosts, as well as phishers, fraudsters and other cybercriminals, can disrupt the attempts of law enforcement agencies by faking their locations.
- **Marketing** – VPN providers would like to sell VPN services in countries where they don't actually own servers.

This project will describe a system for detecting fraudulent IP geolocation – internet hosts using IPs with different country codes to where they are actually located.

In order to explain exactly what IP geolocation fraud is, let us consider a concrete example. *HideMyAss!* is a VPN provider which advertises having servers in many regions of the world, including North Korea [9]. The hostname for one of their "North Korean" VPN servers is `kp.hma.rocks`, which resolves to the IP address of 5.62.61.64. A WHOIS lookup on this IP confirms that the country code of the IP address is indeed North Korea. But if we `ping` this target (measure the time it takes to send a packet to the target and receive one back) from London, we get a latency of 46ms – a one-way delay of 23ms. North Korea is 8,500km away from London, but it takes light 28 milliseconds to travel 8,500km. Thus, it cannot be the case that *HideMyAss!*'s server is in North Korea. This is fraudulent IP geolocation.

A VPN provider using incorrect geolocation might seem relatively benign, but there are many cases where geolocation fraud is more serious. One example encountered during this project is a host serving web content for `coronarelif.org`, a fake coronavirus charity page. If an investigator performed a WHOIS lookup on the IP for this site, the returned record would indicate that the IP is in India. However, this site was actually being served from a server located in the United States.

A particularly prominent example of IP geolocation fraud was discovered in the Tor network. Until recently, the highest-bandwidth exit node in the network was `Hydra1`. This exit node advertised being in Liberia, when in fact the system developed in this project reveals it was located in Sweden, most likely in Malmö or Gothenburg (Figure A.1). This incorrect geolocation may have been used to throw law enforcement investigations off the trail.

The WHOIS protocol [10] is the standard way of finding information about an IP address or domain name. A country code is included on the WHOIS record for IP addresses – and it is the incorrect usage of this country code field that this report will be focusing on. It is trivial to fake this country code – for one Regional Internet Registry, we were able to change the location of a block of IP addresses using a simple online form with no validation. The other Regional Internet Registries are likely using similar systems.

In some cases, it is possible to have the target co-operate with geolocation efforts. For example, Content Delivery Networks get their targets to run measurements to find the closest server. The latencies of these requests could be used to estimate the target's location. Our targets will not be co-operating in this way, so we will instead study *client-independent* geolocation.

Geolocation databases are widely used in industry. Some popular examples include MaxMind's GeoIP, IPData.co and IPInfo.io [11, 12, 13]. These databases are fairly accurate in most cases, but their data is based at least partially on WHOIS data. For example, at the time of writing, MaxMind's GeoIP, IPData.co and IPInfo.io all report `Hydra1` as being in Liberia. Our system is different in that it will not rely on attacker-controlled information.

There has been a great deal of research on theoretical IP geolocation, but applying this theory to detect geolocation fraud in the real world is a less-studied area. One prominent example of a study of this kind is a 2018 paper, *How To Catch When Proxies Lie* [14], which focused on fraudulent geolocation in the context of VPN providers. This study examined 7 VPN providers, and did so in a client-dependent way – signing up to each individual VPN provider, and connecting to VPNs to run measurements. In total, we will examine over fifty times more IP addresses than this study, using a client-independent system.

1.1 Objectives

This project has a number of objectives:

1. Develop a system that can detect when IP geolocation fraud is occurring.
    (a) This system should be able to make automated decisions when possible.
    (b) When not possible, this system should gather additional information to help a human decide whether the IP is using correct geolocation or not.
    (c) This system should be able to process many IPs at once without supervision.
2. Run batches of interesting IPs through this system, and analyse these results.

1.2 Contributions

In the process of completing this project, a number of contributions to the state-of-the-art in geolocation, and in particular geolocation fraud detection, have been made:

1. Development of a system for detection of geolocation fraud.
2. Development of a new geolocation algorithm, Caliper (Section 3.7), as one part of the overall system.
3. Development of a user interface suitable for assisting internet security companies or law enforcement agencies in finding the real location of a given IP (Section 3.9).
4. Analysis of the prevalence of geolocation fraud in the real world on a scale (to the best of our knowledge) never before published in the literature – over 111,000 IPs have been analysed in Chapter 5.

1.3 Ethical Considerations

As part of this project, we will be applying our system to a large number of real-world IP addresses. We have been careful to ensure that we are appropriately considering the ethical implications of performing this research.

There are are number of legitimate reasons why individual internet users would want to appear to be in different locations, using VPN services or by connecting to the Tor network. For example, users may be using Tor for privacy or anonymity reasons, or they may be connecting to a VPN in order to avoid government censorship.

In this report, we will not be revealing the locations of any *users* of location-changing services such as the Tor network or VPNs. This is not possible using the system we design in this report – even if it was, we would not do this for ethical reasons. We will be examining the geolocation of Tor relays and VPN servers, but this will be done without geolocation of any clients, or users, of these services.

We believe that there are positive ethical outcomes from studying the prevalence of IP geolocation fraud, as we do in this report. For example, making VPN and Tor users more aware of where their traffic may be exiting from allows them to make better privacy choices. We will discuss the ethical implications of this project further in the evaluation (Section 6.3).

Chapter 2

Background

Definition: ***geolocation*** [dʒiːoʊloʊkeɪʃən]

the use of technology to identify the geographical location of an internet user

The company uses geolocation to spot whether fraudsters may be trying to transfer money from a new location.

– Collins English Dictionary [15]

This chapter will introduce background knowledge and related work in the area of IP geolocation. The aims of the chapter are threefold. Firstly, Section 2.1 aims to equip the reader with all of the information necessary for understanding the rest of this report. Secondly, Sections 2.2 and 2.3 aim to give the reader an overview of the current state of the literature regarding IP geolocation algorithms. Finally, Section 2.4 aims to give the reader an idea of some usages of IP geolocation algorithms in larger systems, such as the one presented in this report.

2.1 Foundations

2.1.1 Allocation of IP Addresses

IP address allocation is performed in a hierarchical manner, shown in Figure 2.1. The root of the process is the Internet Assigned Numbers Authority (*IANA*) [16]. This organization delegates contiguous blocks of IP addresses (herein *netblocks*) to Regional Internet Registries (*RIRs*), each of which is responsible for a large region of the world. These RIRs then split up and delegate netblocks further to National Internet Registries (*NIRs*), who can then further delegate to Local Internet Registries (*LIRs*). Internet Service Providers are allocated IPs by RIRs, NIRs or LIRs, and then allocate these IPs to end users. A list of RIRs can be found in Table 2.1.

Blocks of IP addresses are often referred to using CIDR notation. CIDR notation expresses a range of IP addresses in the format of `<IP>/X`, where `<IP>` is the start of the IP address range, and `X` is the length of the prefix in bits [17]. For example, `192.168.0.0/16` expresses a network starting at `192.168.0.0` and including any IPs starting with the first 16 bits of `192.168.0.0` - i.e: all IPs from `192.168.0.0` to `192.168.255.255`.

RIR	Region	Established	Designated /8 Blocks
AFRINIC	Africa	2004	4
APNIC	Asia-Pacific	1993	45
ARIN	North America	1997	36
LACNIC	Latin America	1999	9
RIPE NCC	Europe & West Asia	1992	35

Table 2.1: List of RIRs

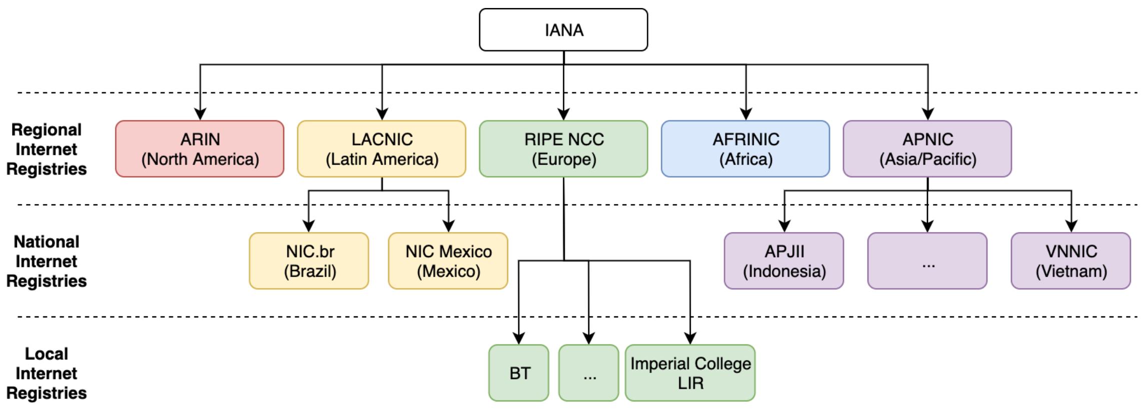


Figure 2.1: Hierarchy of Regional, National and Local Internet Registries. All RIRs are shown, along with a small selection of European LIRs, including British Telecom and Imperial College London.

2.1.2 IP WHOIS

Regional Internet Registries maintain databases containing information about blocks of IP addresses they have allocated – ownership information, abuse contacts, and (importantly for geolocation) country codes. These databases are accessible via queries conforming to the *WHOIS* protocol [10]. The databases return *WHOIS Records* for the netblocks that the IPs being looked up are in.

WHOIS records for netblocks consist of a number of different *objects*, the most important one for our purposes being *inetnum* objects. The inetnum object must contain a country code. As an example, a WHOIS lookup for `155.198.30.71`, an IP address used by Imperial College London, returns information about the `155.198.0.0/16` netblock. The inetnum object of this country code has country code GB.

Unfortunately, it is trivial to spoof this information. For example, RIPE NCC allows the owners of IP ranges to use an online form to set the inetnum country code for their netblocks. No validation is involved in this process. It is this mechanism that allows malicious actors to use fraudulent geolocation for blocks of IP addresses, and this is what this project focuses on detecting.

2.1.3 Fraudulent IP Geolocation

When we use the term *fraudulent IP geolocation*, we are describing the case where a host is in a different country to the country code of the netblock of the IP address that it is using.

2.1.4 DNS LOC

Another source of geolocation information available on the internet is the DNS LOC record. These records allow the owner of a domain to associate that domain name with a latitude, longitude and altitude. Unfortunately, there is no verification system for DNS LOC records, and the domain owner can set them to whatever they would like. This makes them unhelpful for client-independent geolocation, as implemented in this project. Additionally, DNS LOC records are very rarely used. In 2014, Cloudflare reported that out of the millions of DNS records they handled, only 743 were DNS LOC records [18].

2.1.5 Autonomous Systems, BGP and Internet Routing

In order for traffic to move between two hosts on the internet, it must be routed through a series of different intermediate networks and routers. The internet organises these intermediate networks into Autonomous Systems (ASs) - connected groups of IP address prefixes presenting “single and clearly defined" routing policies [19]. To demonstrate, consider the Imperial lab machine gpu11, which has been assigned the public IP 146.169.53.215. This is part of the 146.169.0.0/16 prefix, which is part of AS786 (operated by Jisc Services Limited, an organisation which operates networks for many higher education institutions in the UK).

Each AS communicates with its "peers" (other ASs to which it is directly connected) to learn how to route traffic. BGP is the protocol over which this communication is performed. BGP involves ASs "advertising" the prefixes that they can route traffic towards, either directly, or through their peers. A routing table is then built up, mapping prefixes to the peer which is advertising the shortest path to that prefix.

Let's consider an example of sending data to a host in Auckland, New Zealand (223.165.77.244), from gpu11. Our packet travels through Jisc's network, before reaching one of its "border routers", which then must decide how to route the packet further. This border router will look in its routing table for a prefix containing the destination IP address. The router's routing table indicates that the packet should be sent forward to one of Jisc's peers, AS4637 (Telstra Global). Telstra is a network operator which offers IP transit services across Asia. Telstra receives our packet from Jisc at the London Internet Exchange, and then sends it through routers in its AS, until it reaches a border router in Sydney. This border router then sends our packet to one of its peers, AS134220 (UFONE), a New Zealand-based Internet Service Provider (ISP). Our packet passes through UFONE's network, before being passed on again to the border router for AS45179 (SiteHost New Zealand). SiteHost's AS includes the prefix 223.165.77.0/24. This means that it can directly route traffic to IPs in that range – it does this with our packet, and thus it finally reaches its destination.

When BGP is used between ASs, as described above, this is known as *external* BGP (eBGP). Sometimes, BGP is also used *within* an AS, in order to perform routing between different subsets of the AS's prefixes. This is known as *internal* BGP (iBGP). An observation which is key to this project is that, in the context of eBGP, network operators filter out prefixes of size /24 or smaller. This means that, assuming no iBGP is being used, two IPs in the same /24 block will be in the same country.

2.1.6 VPNs

A VPN, or Virtual Private Network, allows a user to remotely and securely access a different private network. Initially used almost exclusively in business for remote working, they have gained popularity in the last decade for privacy and anonymity purposes. By routing traffic to a different private network, the public IP address of the user can be hidden. Their traffic is *proxied* by the VPN server, meaning the outside world sees the IP address of the VPN server, instead of the user's IP address. IP addresses can be used to personally identify users, which makes hiding IP addresses attractive to privacy-conscious internet users.

Privacy-conscious users are also mindful about which country the VPN server they are using is located in. VPN servers in so-called "Five-Eyes" (or even "14-Eyes"[1]) countries are seen as being less private, due to the internet surveillance infrastructure in place in these countries.

Because of this, VPN providers have an incentive to offer VPN services in non-14-Eyes countries. However, non-14-Eyes countries tend to have less advanced internet infrastructure, and tend to be further away from the customers of these VPN services (meaning their customers will experience slower internet speeds). This leaves VPN providers with a choice between offering fast services in 14-Eyes countries, or offering slower services in non-14-Eyes countries.

Some VPN providers have found a solution to this problem. Simply put servers in 14-Eyes countries, and set the WHOIS country codes for the servers' netblocks to non-14-Eyes countries. A particularly prolific example of this is HideMyAss [9], which advertises services in 190+ countries, but serves many of them from Prague (for European customers) or Los Angeles (for American customers). This will be explored further in the Experimental Results chapter (Chapter 5).

Clearly there are negative ethical implications of this practice. These VPN providers are falsely advertising their services, and misleading their customers. For example, an American HideMyAss

[1]United States, United Kingdom, Canada, New Zealand, Australia, Denmark, France, Netherlands, Norway, Germany, Belgium, Italy, Sweden, Spain

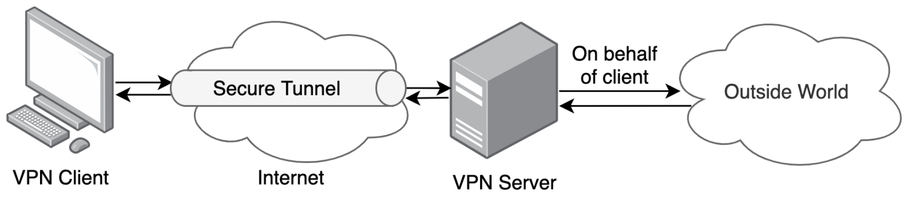


Figure 2.2: Diagram showing the mechanism behind Virtual Private Networks.

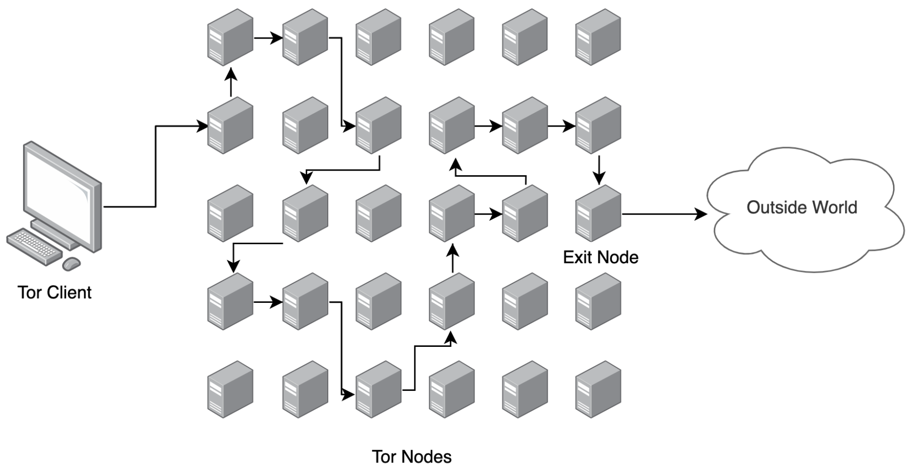


Figure 2.3: Diagram showing the mechanism behind the Tor network.

user could be using a VPN server advertised as being in a non-14-Eyes country, not knowing that their traffic is actually being proxied by a server in Los Angeles, subject to exactly the kind of surveillance they would be under if they weren't using a VPN at all.

When we talk of "geolocating a VPN server", we should be clear that we are only geolocating the VPN server itself, and not any individual users. Geolocating individual users is neither possible, nor would it respect the privacy of users. Geolocating the server itself, and discovering VPN providers who are dishonest about their services, is an action which is positive for the privacy of users.

2.1.7 Tor

Tor, or The Onion Router, works by routing the traffic of the client randomly around a network of *Tor relays* [20]. This process makes it hard for an observer (such as a government intelligence agency) to track which client is making which requests. Requests are routed through the network using the Tor protocol until they reach an *exit relay*, which then proxies the request on to the outside world, as if the client itself was making the request. Responses are received by the exit relay and sent back through the network to the client. In this way, Tor acts somewhat like a VPN, but with many extra layers of misdirection.

There are also incentives for Tor exit relay hosts to use fake geolocation (although these incentives are different to those of VPN providers). Tor is often used for illegal activities, and exit relays are liable for these activities in some countries. Exit relay hosts in countries with these rules have an incentive to advertise their relays as being in countries such as the Seychelles, where the laws (and law enforcement services) are not as strict.

2.1.8 Understanding Network Delay

In order to understand active geolocation algorithms, it is important to understand the nature of network delay. A Detailed Path-Latency Model for Router Geolocation [21] gives a good model for reasoning about the delay from one host to another on the internet. In order for a packet to travel from host A to host B, it must travel on a path through a number of intermediate routers (colloquially known as “hops” in the literature), each connected by a *link* (normally, an optical fibre cable). At each stage of a packet’s journey, delays are encountered – some of them necessary and easily predictable, and some of them harder to predict. There are four types of delay in this model:

- **Queuing Delay** (D_q) - time the packet spends queuing to be processed by routers. This delay will be higher when the packet passes through a highly loaded router.
- **Processing Delay** (D_{pc}) - time the router takes to read the header of the packet and to decide what to do with it. This tends to be a very small delay, unless the router is applying encryption or inspecting packets [22].
- **Transmission Delay** (D_{tr}) - also known as store-and-forward delay, this is the time it takes for the router to push the bits of the packet into the link. This delay is proportional to the size of the packet and the *data rate* of the link.
- **Propagation Delay** (D_{pg}) - time it takes for the signals representing the packet to go from one end of the link to the other. This delay is proportional to the physical length of the link.

Queuing and processing delays are responsible for much of the random variance in latency measurement. Because queuing and processing delays occur at each hop a packet travels through, the more hops there are, the greater the random variance in latency measurements becomes.

Propagation delay is the most important kind of delay for geolocation, because it is proportional to the length of the link, i.e: the distance between the two hops. Thus, the sum of propagation delays across all the links from A to B correlates with the physical distance between host A and host B.

2.1.9 Measurement Tools

There are two important measurement tools that are used in geolocation algorithms – `ping` and `traceroute`.

ping

A `ping` is a measurement of the latency between one host and another on the internet. `ping` measurements are usually performed by sending an ICMP Echo [23, 24] request packet, and recording the time between sending that packet and receiving a reply from the target. There are a number of other ways of performing `ping` measurements, which can be used to circumvent firewalls:

- TCP SYN - these packets will indicate to the target that we wish to initiate a TCP connection. The target will respond with either a SYN-ACK or a RST packet, depending on whether the port the TCP SYN packet was sent to was open.
- TCP ACK - these packets are used to acknowledge sent data over a TCP connection. The target should respond with a RST packet if no connection exists.
- UDP - sending a UDP datagram to a closed port on the target will result in the target sending an ICMP port unreachable response.

traceroute

A `traceroute` creates a trace of the intermediate routers (*hops*) packets encounter on their way between one host and another. Latencies to these hops are also recorded. This is usually performed by sending ICMP echo request packets with increasing time-to-live (TTL) values. If a packet is sent with a TTL of n, the n'th hop will send back an ICMP error message [23, 24]. As such, the n'th hop can be determined by sending an ICMP echo packet with TTL n, and examining the source IP address of the ICMP error message returned. The final hop will reply with a normal ICMP Echo reply, at which point the `traceroute` can be ended. A diagram showing this mechanism can be found in Figure 2.4.

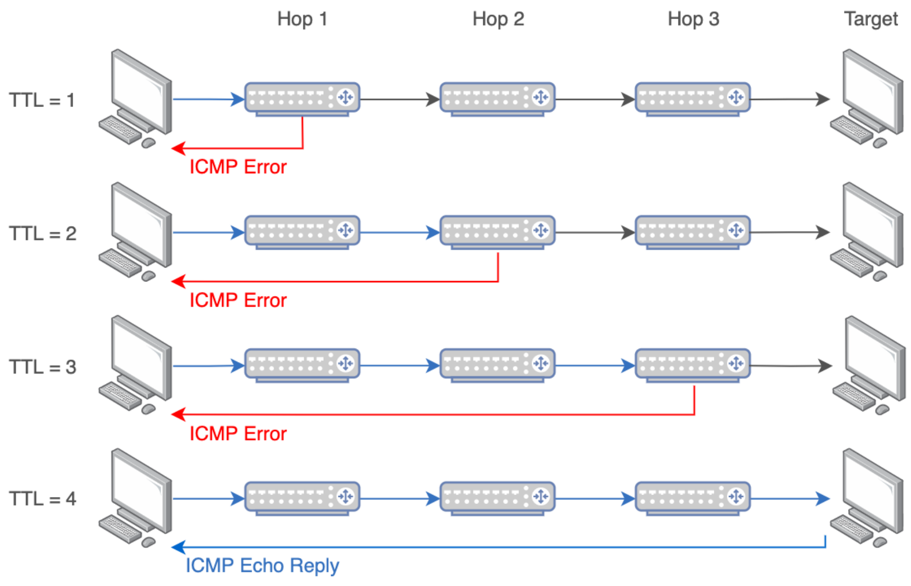


Figure 2.4: Diagram showing the mechanism of a traceroute measurement.

2.1.10 Measurement Sources (Monitors)

A *monitor* is a server which is able to run the measurements detailed in subsection 2.1.9. As will be discussed later in this chapter, often many monitors are required for geolocation algorithms, and the more globally distributed they are, the better accuracy can become. Finding such monitors in geographically diverse locations can be a difficult task, however, a number of organisations do exist to facilitate this.

RIPE Atlas is a network of *probes* (smaller monitors) and *anchors* (larger monitors with higher bandwidth), run by RIPE NCC, one of the Regional Internet Registries [25]. Users who host their own probes and anchors, or sponsor RIPE NCC, are given credits which can be spent to run their own measurements using the network. Measurements can be conducted using the RIPE Atlas API. At the time of writing, there are 660 anchors and over 10,000 probes available in a wide range of locations.

Another network is PlanetLab, a global testbed designed for distributed systems research. Available to research institutions hosting their own PlanetLab monitor, researchers are given SSH access to a range of monitors across the world. Most of the monitors are located at universities. For example, Imperial College is a member of the European division of PlanetLab, and hosts a number of monitors.

2.2 Passive Geolocation

Passive geolocation algorithms perform geolocation without using any active measurements to the target. In the literature, this class of approaches is often also referred to as *data-driven* geolocation, since it relies on publicly available data. Passive geolocation tends to return answers more quickly, however in general the information is of lower quality.

One of the earliest attempts at this class of approaches was **NetGeo** [26], from 2000. Although now defunct, NetGeo was a system run by CAIDA[2], which mapped hostnames, IP addresses and Autonomous System numbers to location data, and allowed the public to query this mapping. NetGeo used two methods to perform this mapping – WHOIS registration record queries, and hostname regexes.

NetGeo performed WHOIS lookups for both IP addresses and hostnames. The information from the returned records was parsed for location data, using a static set of known location keywords.

[2]Center for Applied Internet Data Analysis

A more interesting part of their project was parsing location information from hostnames. In many cases, backbone routers contain location hints such as airport names, presumably in order to make it easy for engineers to quickly identify the location of their routers (see Figure 2.5). The NetGeo researchers manually wrote many regex rules for large internet companies running backbone routers, to extract location information from their hostnames.

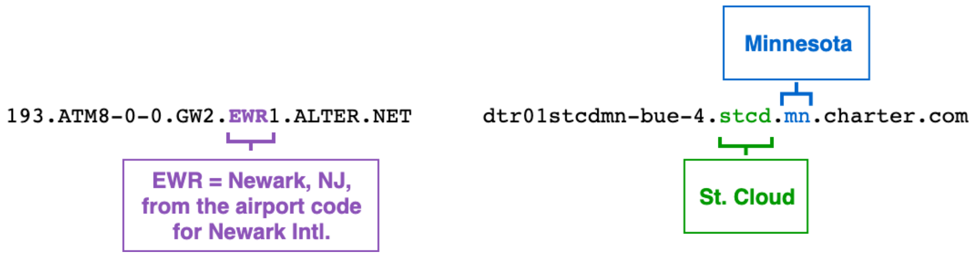

193.ATM8-0-0.GW2.EWR1.ALTER.NET
EWR = Newark, NJ, from the airport code for Newark Intl.
dtr01stcdmn-bue-4.stcd.mn.charter.com
Minnesota
St. Cloud

Figure 2.5: Example of location hints in the hostnames of two American backbone routers.

Because NetGeo is a passive approach, it can return answers very quickly. However manually writing regex rules is time-consuming and tedious. As a result, NetGeo has not been actively maintained for over a decade. The idea of creating regex rules for hostnames was extended by the `undns` tool [27], and also **DRoP** [28].

A state-of-the-art geolocation algorithm using hostname hints is currently being researched by Microsoft [29]. In this paper, a machine learning approach is described. A ground truth set of 67 million IP addresses and locations is mined from the query logs of a popular search engine. A binary classifier is trained on this data, to classify whether a given location candidate is *likely* or *unlikely* for a given hostname.

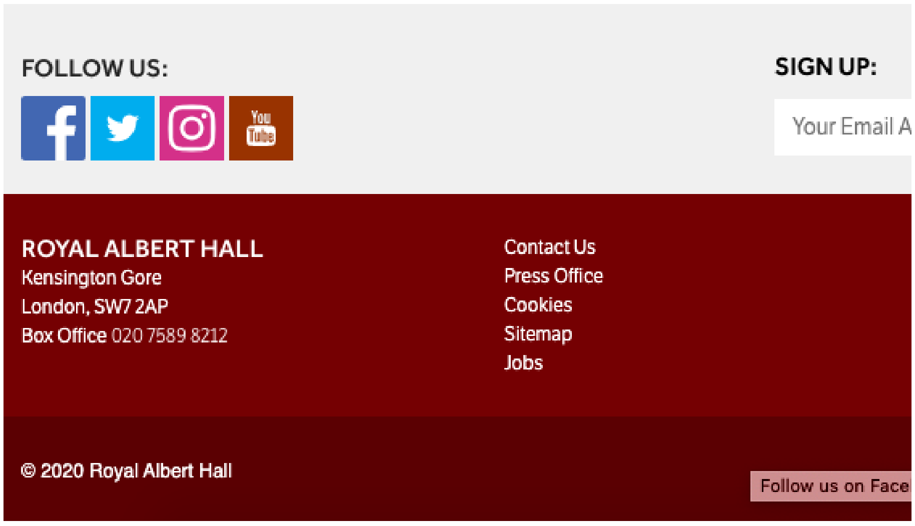
The location candidate is used to generate a list of location hints that may be present in the hostname being considered, if the location candidate is a match for the hostname. For example, a hostname which is a match for “Paris, France” might contain `[paris, prs, fr, par, cdg, ory, bva]`. The classifier uses the presence of these matches as features.

One drawback of the approach in this paper is that it uses reverse DNS to get a hostname associated with the IP, but not all IP addresses have reverse DNS records. A potentially interesting extension would be to `traceroute` to the target, and then perform reverse DNS geolocation on the last hop with a reverse DNS record. An active approach could then be used to constrain the location of the target.

This particular algorithm reports to reduce the median error by an order of magnitude over that achievable using `undns` and `DRoP`. However, the approach requires a large amount of data, the likes of which is only available to large-scale search engines.

Another example of a passive geolocation effort is **Structon** [30]. The paper describes a system which performs web mining to find location data displayed on web pages. An example of this kind of data can be seen in Figure 2.6. The system looked for prefixes such as “Addr:” and “Zipcode:”, in order to find addresses. The location of these addresses is then associated with the IP serving that page. In this way, a database mapping IP addresses to locations is created.

This paper was published in 2007, when the structure of the internet was very different. Locally hosted websites were much more common in these times - AWS was launched just one year before. According to Netcraft, in the period between 2009 and 2013, the number of sites hosted by AWS rose from under 100,000 to over 10 million [31]. This acceleration has continued, to the point where most organisations use hosting providers to host their sites. This poses a problem for Structon’s approach – most of the location data found will be misleading, since most of the sites will be hosted in data centres far away from the business itself.


FOLLOW US:

SIGN UP:

Your Email A

ROYAL ALBERT HALL
Kensington Gore
London, SW7 2AP
Box Office 020 7589 8212

Contact Us
Press Office
Cookies
Sitemap
Jobs

© 2020 Royal Albert Hall

Follow us on Face

Figure 2.6: Example of location data present on the Royal Albert Hall's website.

2.3 Active Geolocation

Terminology

The terminology used in the literature is varied, and sometimes contradictory. For clarity, here are definitions of some terms which will be used in this report.

Target:	the host which is going to be located.
Landmark:	a host with known IP address and location.
Monitor:	a **landmark** which can also be used to run measurements.

Active geolocation algorithms use measurements such as `ping` and `traceroute` to constrain the location of the target. In the literature, these algorithms are also referred to as "measurement-driven". In most of these algorithms, it is assumed that the organisation performing geolocation has access to a set of **monitors** which can be used to perform measurements. All algorithms discussed in this section are *client-independent*.

2.3.1 Nearest Neighbour

One of the simplest active geolocation ideas is to measure the latency between many monitors and the target, and then to use the location of the monitor with the lowest latency ("nearest neighbour") as the location of the target. This is referred to as "Shortest Ping" in [32]. Naturally, this approach is only effective when a monitor exists close to the target, because the minimum error for a given target is equal to the distance from the target to its nearest monitor. As such, a very large number of monitors would be necessary for performing geolocation on a global scale.

A refinement of this idea can be found in **GeoPing** [33]. In this approach, the set of monitors `pings` a set of landmarks, and a "delay vector" is recorded for each landmark - that is, a vector $[D_0, ..., D_N]$ where D_k is the delay recorded by the k'th monitor. A delay vector is then also recorded for the target. Next, the "nearest neighbour" of the target's delay vector is found, by searching for the landmark distance vector with smallest Euclidean distance to the target distance vector. The landmark with this nearest neighbour distance vector is reported to be the location of the host.

The simplicity and reasonable performance of the GeoPing approach mean that it is widely cited and often used as a baseline to compare other algorithms to. However, the drawback is that a large amount of landmarks are necessary to get any reasonable accuracy. Even in the best possible case,

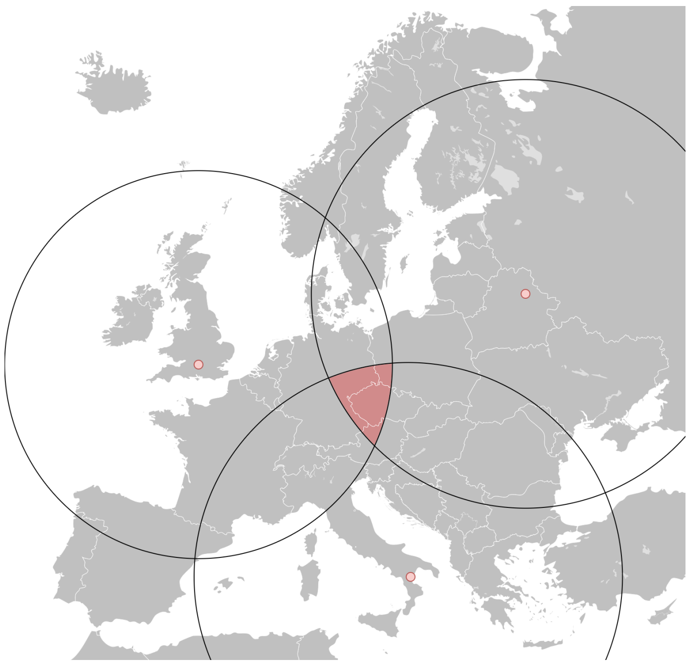


Figure 2.7: Example of performing multilateration using monitors in the UK, Italy and Belarus. The red area shows the range of possible locations of the target.

GeoPing will have an error equal to the distance between the target and the nearest landmark. The difficulty of acquiring such landmarks on a global scale thus poses an issue for using GeoPing globally.

2.3.2 Multilateration

Another key active geolocation approach is to use `ping` measurements to record latencies to the target from many different locations, and then use these measurements to generate "feasible regions" in which the target can lie, given speed of light constraints. The intersection of these feasible regions gives the range of locations in which the target can lie. This approach will be referred to herein as **multilateration**.

Figure 2.7 shows an example of the concept of multilateration. The circles around the monitors show the distances derived from `pings` to the target. These distances are derived using the relationship between round-trip time (RTT) and geographical distance. A common rule of thumb found in the literature is that

$$\mathrm{d}(x, y) \leq \frac{2}{3} \frac{\mathrm{rtt}(x, y)}{2} \mathrm{c}$$

where $\mathrm{d}(x, y)$ is the geographical distance between hosts x and y in meters, and $\mathrm{rtt}(x, y)$ is the round-trip time between hosts x and y in seconds, and c is the speed of light in a vacuum [34, 35]. RTT can be measured easily using `ping` measurements. Conveniently, this relationship gives another rule of thumb – for each millisecond of RTT, there will be roughly 100km of distance.

Of course, this relationship can be refined. For example, in *A Detailed Path-Latency Model for Router Geolocation* [21], a path-latency model is described, where latencies are broken down into their constituent parts. In particular, they separate latencies into processing, queuing and transmission delays, per `traceroute` hop. Additionally, the time taken to generate an ICMP Echo reply is taken into account. These delays can be subtracted from the overall latency, in order to more closely model the actual latency between the hosts. By more closely modelling the latency, a tighter bound on distance can be determined.

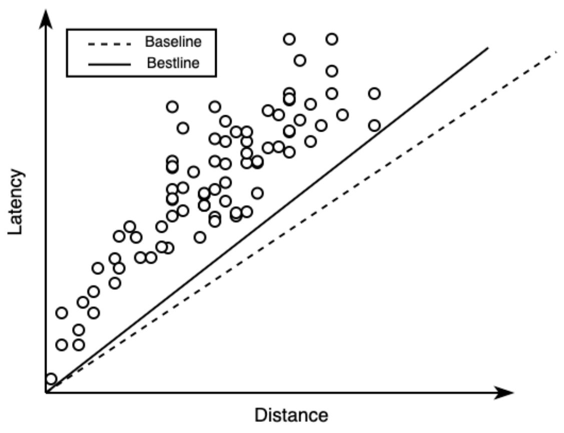


Figure 2.8: Example of the concept of fitting a bestline to latency-distance measurements, as described in [36].

This approach involves using `traceroute` probes, which is inherently more complicated and expensive than using simple `ping`s. Additionally, they advocate for making blanket assumptions about delays (for example, $100\,\mu s$ for the echo packet generation) which may not hold in all cases, meaning distances could be underestimated.

Constraint-Based Geolocation

Another approach to refining the latency-distance relationship is **Constraint-Based Geolocation (CBG)** [36], developed in 2006. This paper describes a system for calibrating the latency-distance relationship. This is done by measuring the latency between all monitors in the geolocation system. All of these monitors have a known location, so a ground-truth set of latency-distance measurements can be generated. A "bestline" is then fitted to this data. An example can be seen in Figure 2.8.

The "bestline" attempts to improve the $\frac{2}{3}$ rule-of-thumb "baseline", by fitting a line with gradient such that no latency-distance pairs falls below the line. This line can then be used to map latencies to maximum distances, in place of the "baseline" conventionally used. Multilateration can then be performed with these distances, which should now reflect the actual distance to the target more accurately. The paper reports significantly better geolocation accuracy than using basic multilateration.

CBG is used as the basis for many subsequent algorithms, and like GeoPing is often used when comparing the performance of other geolocation algorithms. It is not perfect, however. The use of the bestline relationship can be too optimistic in some cases. Weinberg et al. [14] report that network congestion when the monitors are calibrated can lead to distance estimates being too low. This can cause mispredictions, and, even worse, failure to make any prediction of the location at all (if the feasible regions from each monitor do not intersect due to underestimation of distance).

Topology-Based Geolocation

Topology-Based Geolocation (TBG) [32] is an extension of CBG, also published in 2006. The researchers found that while delay-based algorithms work well when monitors exist close to the target, they perform poorly when this isn't the case. TBG attempts to mitigate this problem, by considering the *topology* of the network. In particular, they simultaneously geolocate intermediate hops to the target, discovered via `traceroute`, using the inter-hop latencies to further constrain the positions of all the hosts considered. All of these constraints are combined to create an optimisation problem, solvable using a semidefinite program solver such as SeMuDi [37].

TBG also uses *interface clustering*, to identify intermediate IP addresses which belong to the same router, on different interfaces. This allows them to more tightly constrain the location of routers, because all of the separate constraints for the individual interfaces can be used in combination, in the knowledge that the interfaces are in fact are located in the same place. The Ally [38] and Mercator [27] tools are used to perform this interface clustering.

The TBG paper also reports a number of extended variants, which are reported to produce even better results. The first of these is **TBG-Passive**. This variant uses landmarks to generate more constraints on intermediate routers. The second is **TBG-undns**, which additionally uses location hints identified in intermediate hop hostnames. Location hints are verified before use, by making sure that they don't violate speed-of-light constraints.

Results reported in the TBG paper indicate that it achieves better results than CBG. The best-performing variant was TBG-undns. There are some disadvantages of the TBG approach, however. Firstly, `traceroute` measurements are more time-consuming than regular `pings`. Secondly, it departs from the simple geometrical approaches considered before now, and uses optimization to find a solution. This is a considerably more computationally expensive process.

Octant

Another extension of CBG is **Octant** [39], which introduces the idea of *negative constraints*, which describe areas in which the target cannot lie. First, similar to CBG, latency measurements are performed to hosts with known location. A convex hull is then fitted to the set of latency-distance measurements, so that there exists a function giving the minimum distance for a given latency, and a function giving the maximum distance for a given latency. A diagram of this can be found in Figure 2.9.

Using this, a latency measurement for a target can be transformed into an *annulus*[3] in which the target is likely to lie, rather than a disc as in CBG. These regions are represented using Bézier curves. Octant uses efficient geometrical computations using these curves to reach a solution, making their framework less computationally expensive than an algorithm relying on optimization to find a solution such as TBG.

Another interesting contribution by the authors of the Octant paper is the idea that not all latency measurements are of equal value. The researchers found that introducing a system where lower latencies are weighted higher improves performance by approximately 33%. This, they believe, is due to paths with higher latency being less direct, and more often involving congested links. Additionally, they found that the latency-distance relationships derived using the convex hull technique are not conservative enough at higher latencies, so they use a looser speed-of-light based relationship above a certain latency threshold.

Octant can also extract information about intermediate routers and use them as landmarks when locating the target. In this mode, the Octant algorithm is run in full against each intermediate hop. `undns` is also used to extract location data from hostnames.

Finally, Octant is able to incorporate geographical and demographic information such as population density data, and information on uninhabited regions of the Earth, in order to provide more negative and positive constraints.

Octant is considered to be one of the best-performing CBG-based geolocation algorithms. However, there are some disadvantages of their approach. Excellent results are reported in their paper, but analysis is limited to North American targets. They use 50 monitors for these experiments, which is a large number in comparison to other approaches, especially given that they only focus on one continent. This number would presumably need to be increased even higher for their approach to give similar results on a global scale. Also, the complexity of their algorithm means that Octant has not (to the best of our knowledge) been directly evaluated against newer algorithms in the literature. Researchers who do make comparisons usually use some kind of "Octant-based" approximation, such as in Eriksson et al. [40].

[3]a ring-shaped region, formed from subtracting a smaller concentric circle from a larger one

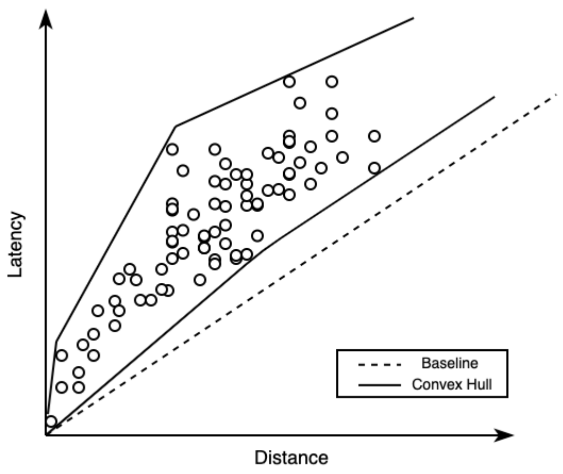


Figure 2.9: Example of determining maximum and minimum latency-distance relationships, using a convex hull approach similar to that described in Octant. Data is the same as in Figure 2.8.

2.3.3 Shared Path Geolocation

One of the more recent contributions to the geolocation field is the concept of *Shared Path Geolocation*. The idea is that `traceroute` measurements can be run on many landmarks, and the path taken to those landmarks can be stored, along with the latencies between hops. Then, when a target is being geolocated, the path of packets to the target is inspected. If this path shares hops with a path seen earlier, the latencies between the *last shared hop*, the landmark and the target can be used to constrain the location of the target.

Towards Street-Level Client-Independent Geolocation [41], commonly referred to as Street-Level Geolocation or **SLG**, used this approach, along with a web mining approach for generating landmarks similar to that described in the Structon framework. First, they used CBG to narrow down the possible location of target. Next, they used online mapping services with queries such as “business”, “university”, “government office”, constraining results to certain zip codes, in order to find sites hosted on-premises at businesses near the target. The servers hosting these sites could then be used as landmarks. The landmark with minimum estimated latency to the target (estimated by taking the sum of the delay from the closest common router, as displayed in Figure 2.10) is taken as the estimated location of the target.

SLG was published in 2011, during the rise of cloud hosting. Because of this, many of the sites they scraped location data from would have been hosted by cloud providers. Thus, they needed to employ methods of landmark verification – verification that the site is actually locally hosted, rather than in a data center. One method used was to access the candidate landmark site through both its hostname and its IP address. If the candidate site is served through a CDN, or hosted on a shared hosting platform, many sites will be using the same IP address as it, so it will not be accessible through its IP address. Conversely, a site being accessible through both its IP and hostname is a strong indicator that it is hosted locally.

Unfortunately, this landmark verification scheme would likely not work in the context of today’s internet. Sites following good security practices and serving only over HTTPS will not be accessible via their IP address, since SSL certificates are issued for domain names and not IP addresses. The SLG paper was published in 2011. Google reports that in 2015 only 44% of traffic for US Chrome users was being served through HTTPS, but this figure has now risen to 95% [42]. This implies that finding non-HTTPS web servers to verify in this way would be significantly harder in today’s internet.

On top of mining the web for landmarks, SLG used the query logs of a mapping service to extract mappings between IP addresses and locations. For example, if there are many queries originating from an IP address searching for directions from Fulham to other locations, it could be assumed that that IP address is located in Fulham, London. The SLG paper does not report how many landmarks were used, and it also limits its analysis to US-based targets. As such, it is unclear how many landmarks would be needed on a global scale.

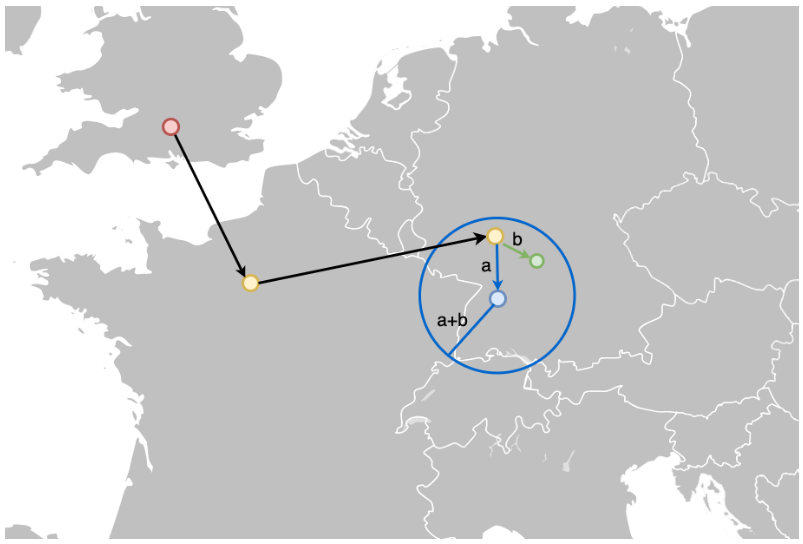


Figure 2.10: Diagram showing the mechanism of shared path geolocation. A UK-based **monitor**, shown in red, has run a traceroute on the blue **landmark**, using the yellow intermediate hops. When later locating the green **target**, the latencies from the last shared hop to the landmark and the target (a ms and b ms respectively) can be added, to generate an extra constraint on the target's location.

A further study into the idea of shared path geolocation was conducted in 2019 by Wang et al. [43] (this approach will be referred to herein as **Common Router Geolocation**). This approach is similar to SLG in that it requires mining landmarks from the web (and indeed their strategy for mining landmarks is very similar to SLG's). However they use these landmarks in a different way – they aim to find and geolocate "common routers", which can then be used when geolocating targets. Common routers are found by tracerouting landmarks and finding IP addresses which occur in multiple traceroutes. Their location can then be estimated by using estimated latencies between them and landmarks. As an example, in Figure 2.10, the final yellow router could be used as a common router, if both the green and blue hosts were discovered as landmarks. The latency between the yellow router and the green landmark would be estimated as the latency from the red monitor to the green landmark, minus the latency from the red monitor to the yellow router. These latencies can be used to constrain the position of the router using (for example) Constraint-Based Geolocation, or indeed any other latency-based geolocation algorithm (a number are discussed in Wang et al.'s report [43]).

When geolocating a target, a traceroute is performed to the target. Common routers identified earlier are then extracted from the list of hops in the traceroute. The latency between the target and the common router is estimated as the latency to the target, minus the latency to the common router. We then have extra information that can be used to constrain the location of the target.

The paper reports a 10% improvement on the error obtained by Street-Level Geolocation. However, the approach described in the paper suffers from the same issue discussed with SLG – it uses differences between IP and domain name access to disambiguate locally hosted from cloud hosted websites, which is not possible for HTTPS-only sites. The study focuses only on China, where HTTPS adoption is very low compared to the rest of the world – research produced by Google and Mozilla indicates that China has the lowest HTTPS adoption of any country in the world, excluding countries with low internet users [44]. This means that while the IP access location verification scheme worked in China, it may not work on a global scale.

2.3.4 Statistical Geolocation

The newest frontier of geolocation algorithms rely on statistical methods. These algorithms tend to have increased accuracy, while requiring less monitors. However, they also tend to be more computationally expensive.

One of the first statistical approaches to geolocation was **Statistical Geolocation of Internet Hosts** [45]. In this approach, instead of fitting bestlines or convex hulls to latency-distance measurements, Kernel Density Estimation is used. As in previous algorithms, the latency from each monitor to a set of landmarks is measured. This time, however, each monitor will use a kernel density estimator, trained on the set of monitor-landmark latencies, to find the likelihood of a given latency-distance pair being correct.

Then the probability of a candidate location being correct is defined as such:

$$P(c \mid L) = \prod_{m \in M} P_m(\text{distance}(c, m) \mid L_m)$$

where M is the set of all monitors, L is the set of latencies such that L_m is the latency measured from monitor m to the target, and $P_m(d \mid l)$ returns the probability of the target being distance d from the monitor, given the observed latency l, derived from the KDEs discussed earlier.

The argmax of this function is then found, using an optimization method, yielding the estimated location of the target. The paper reports that this method achieves mean error 35% lower than CBG.

In 2010, **A Learning-based Approach for IP Geolocation** [46] was published by Eriksson et al. In this paper, IP geolocation is remodelled as a machine learning classification problem. A Naive Bayes classifier takes latency measurements, hop counts and population density data as inputs, and returns an estimate for the county in which the target lies.

One Kernel Density Estimator is used to learn the relationship between latency and distance, and another is use to learn the relationship between hop count and distance. The likelihood of the target being in a certain county in the US, given measured latencies and hop counts from a number of monitors, can then be derived by taking the sum of two KDEs, along with a likelihood value derived from that county's population density. The contributions of each source are weighted, and optimal weights are learned using the training data (a ground truth set mapping latencies and hop counts to counties).

The argmax of this likelihood function is used to find the most likely county for the target, given the hop counts and latencies to the target. This is much more computationally expensive than using simple geometric calculations, however the paper reports improvements on CBG in 96% of cases.

A further publication by Eriksson et al. is **Posit** [40], from 2012. Posit is the successor to *Statistical Based Geolocation of Internet Hosts*, discussed earlier. It learns the latency-distance relationship in the same way described in that paper. However, it augments the method by adding the use of landmarks. The key observation is that even though the landmarks cannot be controlled, the distance between a target and a landmark can be inferred using only measurements from monitors to the target and the landmark.

In particular, on top of the monitor-target likelihood function $P_m(d, l)$ as in [45], they learn a likelihood function $P_l(d, v)$, where d is the distance between a target and a landmark, and v is the "threshold L1-distance" between the two, a metric calculated using the latency vectors measured to the landmark and target. The relationship between threshold L1-distance and geographical distance is learned using kernel density estimation with a set of training targets.

Then, to find the location of a target, the sum of the monitor-target likelihood function and the target-landmark likelihood functions is maximised. The region of candidate locations to optimize over is found by first running CBG.

The researchers report using 25 monitors and 75 landmarks, and they report a median error 60% less than all methodologies they tested against (including CBG, GeoPing and an Octant-based approach).

All of the statistical-based geolocation algorithms discussed rely on searching a space of possible solutions to find the most likely one. This makes them more computationally expensive than other approaches. However, they also report achieving better results than purely multilateration-based algorithms. This is probably because they are able to better approximate the latency-distance relationship, and in Posit's case because they are able to make better use of landmarks.

2.3.5 Comparison of Active Geolocation Algorithms

Algorithm	Year	Geographical Scope	Complexity	Monitors Used	Error (km)
GeoPing [33]	2000	US	ping-only	$\sim$10	Median <500
Path-Latency [21]	2009	Europe	traceroute	18	Mean <150
Constraint-Based [36]	2006	Western Europe	ping-only	42	Median 25
Topology-Based [32]	2006	Europe	traceroute	68	Median 67
Octant [39]	2007	North America	traceroute	53	Median 35
Street-Level [41]	2011	US	traceroute	1 *	Median 2.11
Statistical Geolocation [45]	2009	US	ping-only	85	Median 53
Learning-Based [46]	2010	US	ping-only	78	Median 253
Posit [40]	2012	US	ping-only	25	Median 44

Table 2.2: Comparison of the geolocation algorithms described
* the number of monitors used is not explicitly mentioned, but the description of the algorithm suggests only a single monitor is necessary.

Table 2.2 displays a comparison of the characteristics and reported results of the active geolocation algorithms discussed above. The varying quantities of monitors and data used to produce results makes it difficult to compare between the results of different algorithms. For example, Street-Level geolocation appears very attractive, but their results are predicated on them receiving an undisclosed amount of data from an online mapping service. As another example, the Path-Latency results look poor in comparison to other algorithms, but they used a significantly lower number of monitors in their analysis. For this reason, when evaluating Caliper, a like-for-like comparison to other geolocation algorithms (using identical numbers of monitors and landmarks) will be performed.

None of the papers discussed in this section report results on a global dataset, as will be discussed in this report. Caliper, the new geolocation algorithm which will be presented as part of this report, will thus be somewhat unique in that it has been designed with performance on a global scale in mind.

2.4 IP Geolocation in Practice

A number of studies have discussed the usage of IP geolocation algorithms in practical settings, and in larger systems such as the one described in this report. In this section, we will discuss some of these, and discuss the lessons we can learn from them. We will focus particularly on the applications of IP geolocation in the field of cybersecurity.

2.4.1 Geolocating Fast-Flux Hidden Servers

One such application which has been described in the literature involves finding the geolocation of a malicious server hidden behind a botnet using the fast-flux DNS technique. A botnet is a collection of computers, usually geographically dispersed, which have been infected with malware and thus can be made to carry out acts on behalf of the owner of the botnet. Fast-flux is a DNS technique used to hide the address of malicious servers, using botnets. The DNS record for the malicious site will be changed at a high frequency. The key technique is that these DNS records will not point to the malicious server itself – they will point to computers in a botnet, which will act as proxies in front of the malicious server. This technique obscures the actual address of the malicious server [47].

The first study in this area was conducted by Castelluccia et al. [48]. In this study, botnets hiding the addresses malicious sites hosting web content were examined. Because the malicious sites were serving web content, the botnet proxies would receive HTTP requests and relay them to the malicious server. Thus the researchers performed HTTP requests to the malicious site, and timed the responses in order to estimate the latency to the malicious server itself. They performed these measurements from a number of monitors, and through a number of botnet proxies for each monitor, taking the minimum latency for each monitor. This was in order to reduce the latency inflation due to indirect paths taken by packets (if the bot proxying traffic is far from the actual malicious server, this would inflate the measured latency considerably). They then used Constraint-Based Geolocation (discussed in Section 2.3) with these latencies in order to determine the region in which the malicious server could lie.

MISHIMA, another study performed concurrently to the one performed by Castelluccia et al., offers a different approach [49]. The authors of MISHIMA claim that its geolocation capabilities exceed that of the system presented by Castelluccia et al. MISHIMA first aims to geolocate the proxies themselves, using multilateration from many monitors (“approximately 150-200 PlanetLab nodes”). Then, latency measurements are recorded by timing HTTP requests. The latencies between proxies and the malicious server are estimated as the latency to the malicious server, minus the latency to the proxy. These latencies are then used to perform multilateration, with the result being the region that the malicious server could be located within.

The MISHIMA paper also describes a process for finding the IP address of the malicious server, given the geographical region that it lies within. This is important, since finding the IP address of a malicious server is necessary for it to be taken down. This lookup is performed by searching a graph generated from the CAIDA Macroscopic Internet Topology Data Kit (CAIDA ITDK), which enumerates blocks of IP addresses and their geolocation. The CAIDA ITDK uses MaxMind’s GeoLite City database to generate this geolocation information. As discussed earlier in the report, MaxMind databases use netblock country codes as one component of their geolocation process, meaning their data is (at least in part) vulnerable to the kind of geolocation fraud discussed in this report. Thus, it is feasible that an organised attacker could purchase a block of IP addresses, set its inetnum country code to a different country, and use an IP from this block for the malicious server, in order to avoid detection by MISHIMA.

2.4.2 Location-Based Server Authentication

In 2017, Abdou et al. published a study into the use of geolocation to augment TLS authentication of servers being connected to by clients [50]. As well as the client verifying the certificate of a server, the researchers advocate the client additionally verifying the physical location of the server. They argue that this adds a layer of resistance to MiTM attacks involving the compromise of Certificate Authorities.

They suggest that a central system could perform geolocation verification, with clients making requests to this central system when making initiating TLS connections. In the model presented, servers “assert” their location, which is then “verified” by the central system. A network of monitors (in the paper, PlanetLab servers were used) is used to perform this verification. Monitors are selected in threes, such that they form triangles encapsulating the server’s asserted position. Thales’ theorem is used to determine whether the host being geolocated could be within the triangle. The browser can then choose to drop connections or warn the user if the location verification fails.

The researchers extend this idea by describing a system of location pinning, using an idea analogous to TLS certificate pinning. Under this system, browsers would build up a mapping between servers and their expected locations. Building up this mapping could take place in three ways:

1. Installing into browsers a preloaded list of locations for particular servers;
2. Receiving instructions from the server to pin particular locations (analogous to HTTP Public Key Pinning for TLS);

3. Building up a list of locations for servers as they are visited by the user.

This scheme could increase the quality of authentication of TLS connections, but it has not yet been adopted anywhere, or discussed further in the literature. One potential issue with the system is that the central system could become a single point-of-failure.

We can learn some lessons from their approach to building an IP geolocation system as a whole. In particular, the researchers describe caching the results of lookups in order to increase the efficiency of their system. They describe caching these results for relatively long periods of time (at least hours). Their idea of having a central server responding to queries while controlling monitor nodes will also be relevant when designing our geolocation system.

2.4.3 How To Catch When Proxies Lie

How To Catch When Proxies Lie [14] is the only existing study examining the prevalence of IP geolocation fraud in the real world that we could find. This study looks into the geolocation used by 7 (unnamed) VPN providers.

The paper begins by identifying the problem with relying on commercial IP geolocation services, which themselves rely on attacker-controlled information such as WHOIS information and router hostnames. The authors note the importance of VPN providers being honest about their geolocation – the authors were compelled to perform this study when they attempted to use VPN services to perform censorship monitoring, but found that they weren't able to observe censorship when connected to a VPN allegedly in the censoring country.

The researchers go on to describe the approach used to get the results presented in the paper. They used a client-dependent approach – they connected to VPNs and then sent measurements through them to hosts with known location. These measurements were then used to perform active geolocation, using a modified version of Constraint-Based Geolocation. This approach is different to the one used in this project, since we will be performing measurements *to* targets, and not the other way around.

Interestingly, the researchers found that Constraint-Based Geolocation, a relatively simple algorithm, was able to outperform more complicated algorithms, such as their Octant approximation. The more complicated algorithms often gave answers which did not include the actual location of the target, which is particularly problematic when assertions are being made about the honesty of targets. This is a useful piece of information for us to consider when designing our system.

2269 IP addresses were considered in this study, and the researchers were able to prove that 638 out of these addresses (28.1%) were using incorrect geolocation. We will study a similar set of VPN server IP addresses later in this report, using our client-independent system. Because our system is client-independent, we will also be able to study a far larger and more diverse set of internet hosts, including web servers, Tor nodes, and VPN servers.

Chapter 3

Design

In this chapter, we will discuss our fraud detection system. We will start by describing the overall design of the Fraud Detection Pipeline, which is the core of our system, and then we will provide further details on each individual stage of this pipeline. After that, we will discuss the interfaces through which our system can be interacted with. We will finish by explaining the design of the software deployed on our measurement servers.

3.1 Fraud Detection Pipeline Design

This section will discuss the Fraud Detection Pipeline – the core of our fraud detection system. This pipeline will receive IP addresses, perform measurements and analysis on the IPs, and then return information about their geolocation.

When possible, the system will return a conclusive answer about whether the IP has fraudulent geolocation. However, geolocation algorithms are imprecise and we cannot rely on them always producing conclusive answers. As such, the system will also return a number of other pieces of information, which can assist a human in determining the validity of the IP's advertised geolocation.

In order to perform geolocation fraud analysis on a target, resources are needed – time, compute power, network bandwidth, and, by proxy, money. Our system attempts to minimise the number of resources needed. This is especially important when using credit-rationed resources, such as ping measurements from the RIPE Atlas network. With this in mind, the system designed consists of a number of stages, shown in Figure 3.1.

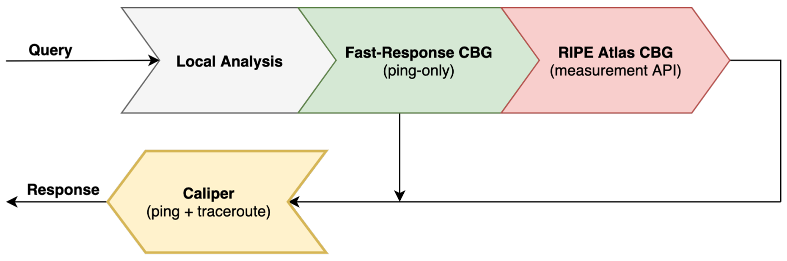


Figure 3.1: Diagram showing the four stages in the geolocation fraud detection pipeline.

The first stage ("Local Analysis") involves performing measurements and analysis which can be conducted locally (on one machine).

The second stage ("Fast-Response CBG") involves running Constraint-Based Geolocation using measurements from an array of measurement servers in our control. Performing measurements from these servers will be quicker than making calls to the high-latency RIPE Atlas API. Constraint-Based geolocation, discussed in the Background chapter, is a computationally inexpensive and reliable geolocation algorithm.

The third stage ("RIPE Atlas CBG") involves running Constraint-Based Geolocation using measurements from the RIPE Atlas network. The RIPE Atlas network allows measurements from many monitors all over the world, but their API has latency in the range of minutes. Therefore it will only be run if the Fast-Response CBG stage was inconclusive.

The final stage ("Caliper") involves running the novel geolocation algorithm developed in this project. Caliper is a statistical geolocation algorithm built with global scale in mind. This stage will be used to generate a "best guess" for the location of the target. This stage does not make conclusive decisions about the location of the target – these decisions are left to the CBG-based previous stages. This is because CBG allows us to definitively state that the host *must* be within the returned region, but a statistical algorithm like Caliper is not as certain.

Conclusiveness

A stage is conclusive for a given IP address if that stage is able to assert that the IP address's WHOIS country code is correct or incorrect. For example, the Fast-Response CBG stage will determine a region in which the IP must be located within, given constraints learned from training data. If this region contains *only* the country the IP claims to be in, then we conclude that it is using correct geolocation. If this region *does not* contain the country the IP claims to be in, then we conclude that it is using incorrect geolocation. If neither is true, the stage is inconclusive.

Design Justification

The design of each individual stage in the pipeline has been chosen carefully. The Fast-Response CBG stage acts as a gatekeeper for the RIPE Atlas CBG stage, to reduce the number of IPs that require expensive and slow RIPE Atlas API measurements.

We have chosen CBG for the stages which make conclusive decisions because CBG is simple and reliable. CBG relies only on very simple latency-distance rules, which essentially approximate the physical limitations of the speed of light in fibre. More complex algorithms can achieve lower median error, but their use of extra (potentially attacker-controlled) information such as traceroute hostname hints introduces uncertainty and the possibility of their guesses not including the target's actual location. We would rather give an imprecise guess, than give a guess that is outright wrong, since an outright wrong guess could lead to us incorrectly classifying targets as using incorrect geolocation. *How To Catch When Proxies Lie* [14] discusses this problem in more detail, and also solves it by using the simpler Constraint-Based Geolocation approach. Caliper uses extra information such as traceroute hostname hints, but it is only used to give a best guess at the target's location, and not for making conclusive statements about the validity of the target's claimed geolocation.

3.2 Sourcing Ground-Truth Data

Both CBG and Caliper require training data, so it is important to identify a source of ground-truth data - i.e: IP addresses with known geolocation. We will also use validation and testing for the evaluation of Caliper. In accordance with machine learning conventions, the term *training data* will be used to describe ground-truth data used to train models. The term *validation data* will be used to describe data used to evaluate the relative performance of different models, in order to select the best possible parameters. The term *test data* will refer to data used to evaluate the performance of the final model.

A varying amount of ground-truth data is used in the literature when developing geolocation algorithms. A comparison can be seen in Table 3.1. As shown in the table, many influential and widely cited studies, such as Constraint-Based Geolocation, Octant and Posit, have used very small testing datasets. Finding a large source of high-quality ground-truth data has been reported as being challenging in many papers in the literature.

None of the existing studies have used validation data to tune model parameters, however Caliper will use validation data to tune its hyperparameters.

Algorithm	Region Considered	Training Size	Validation Size	Testing Size
GeoPing [33]	US	265	-	181,246 †
Shortest Ping [32]	US	-	-	8,321
Path-Latency [21]	Europe	-	-	1,192
Constraint-Based [36]	US + Western Europe	137	-	137
Topology-Based [32]	Europe	-	-	8,321
Octant [39]	North America	104	-	104
Street-Level [41]	US	-	-	160
Statistical Geolocation [45]	US	85	-	85
Learning-Based [46]	US	3375	-	13,350 *
Posit [40]	US	75	-	431

Table 3.1: Comparison of ground-truth data used for evaluation and training in the literature.
† - data was sourced from website user data, including that of Hotmail.
* - ground-truth data was derived from MaxMind lookups on IP addresses.

Region	Count
South America	192
North America	749
Africa	178
Asia	268
Oceania	279
Europe	4224

Table 3.2: Counts of responsive RIPE Atlas probes in each geographical region.

The RIPE Atlas network, described in the Background section, has an extensive list of probes with accurate location data. At the time of writing, there are over 11,000 connected probes, in 176 different countries. A list of these probes, along with their location as latitude and longitude, can be found on `ftp.ripe.net`. This source of ground-truth will be used throughout this project to train, evaluate and test the models being used.

Of these connected probes, 5,903 were responsive to ICMP Echo requests (pings). An important note is that the RIPE Atlas probe location distribution is heavily skewed towards Europe and (to a lesser extent) North America. As this system should be globally applicable, it makes sense to test it with data that is globally distributed. If this wasn't done, the error rate of the system could be misleadingly low – for example models could "learn" to classify targets as being in Europe too often, and be rewarded for this due to the validation data distribution favouring European probes.

To solve this problem, the probe list has been segmented roughly into global regions – South America, North America, Africa, Europe, Asia and Oceania. Each split will have an even number of probes from each of these regions. Table 3.2 shows a list of each region, along with the number of responsive RIPE Atlas probes in that region. The region with the lowest number of probes is Africa, with 178 probes. This means that, in total, there will be $178 \times 6 = 1068$ evenly distributed probes available.

70% of the 1068 probes will be used for testing, a small number (90) probes will be used for validation, and the rest will be used for training. This number of testing instances is competitive with the best testing datasets used in the literature, with the exception of GeoPing and Learning-Based Geolocation[1]. It comfortably exceeds the number of testing data points used for some of the most widely cited studies such as Constraint-Based geolocation, Octant and Posit.

This data will be used when evaluating the performance of Caliper, but also when training both Caliper and CBG for use in the fraud detection pipeline.

[1]GeoPing uses a large amount of data sourced from the users of online websites. Learning-Based Geolocation uses MaxMind to label its testing data, which is undesirable as MaxMind itself is inaccurate.

3.3 Pipeline Caching

As discussed in the Background section, except in the context of iBGP, two IP addresses in the same /24 block will be in the same physical location. Therefore we can cache the results of each stage for /24 blocks. When an IP finishes a stage, the results of that stage will be stored in a cache, along with the /24 that that IP is within. Future IPs in the same /24 will then receive this cached entry, instead of requiring additional unnecessary processing.

3.4 Local Analysis Stage

There are a number of useful pieces of geolocation data that can be determined using only a single machine. The local analysis stage will collect these pieces of data, and will run before all other stages. This section will explain the data collected in this stage.

The most important piece of information collected by this stage is the "advertised country" of the IP address - the inetnum country code of the netblock in which the IP address lies. This is the country code returned on a WHOIS lookup to the IP, and is the country which we will compare against when deciding whether the IP address is using fraudulent geolocation data or not.

3.4.1 Netblock & Parent Netblock

As discussed in Subsection 2.1.1, IP addresses are allocated in blocks in a hierarchical manner. It can be useful to compare information regarding the netblock that the target lies in, with details regarding the "parent" of this netblock (where "parent" of a netblock refers to the smallest larger netblock containing the netblock). For example, if the parent netblock of a Seychelles /24 is a United States /21, this suggests that the advertised geolocation of the /24 netblock may be false. There are situations in which mismatches between parent and child netblock country codes do occur legitimately, but the mismatch tends to be between neighbouring countries, and not countries in different continents.

The following pieces of information will be collected and returned:

- Netblock Address Range
- Netblock Name
- Netblock Country Code
- Parent Netblock Address Range
- Parent Netblock Name
- Parent Netblock Country Code

3.4.2 Regional Internet Registry

The Regional Internet Registry in control of the netblock in which the target lies can also be a useful piece of information to consider when evaluating the veracity of a geolocation claim. For example, if a netblock has been assigned through RIPE (the RIR for Europe, West Asia and former USSR states), but its country code is Hong Kong, this is suspicious. Netblocks actually located in Hong Kong are more likely to be assigned through APNIC (the Asia-Pacific RIR).

3.4.3 Preceding Hop

Knowing information about a preceding hop (i.e: a hop that packets were routed through before reaching the target) can indicate geolocation fraud in some cases. For example, if packets travel through a Czech router immediately before reaching a server advertising to be in North Korea, this indicates that either the router or the target is using fraudulent geolocation – most likely the target.

In order to make the preceding hop information as useful as possible, the closest replying hop to the target will be used. Additionally, selected preceding hops must be less than 3 hops away from the target. The following information about the closest possible preceding hop will be collected and returned:

- IP Address
- Hostname
- Country Code

3.5 Fast-Response CBG Stage

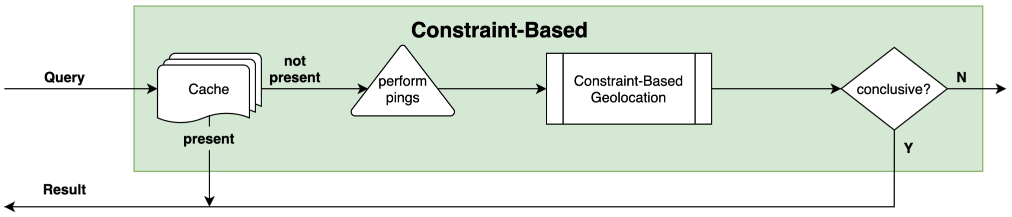


Figure 3.2: Diagram showing the Constraint-Based stage of the system.

This stage involves running Constraint-Based Geolocation (CBG) using measurement servers in our control, running our own ping API. The stage exists to limit usage of the RIPE Atlas stage to IPs that need it – the RIPE Atlas stage is expensive and slow in comparison. The CBG algorithm is shown in Algorithm 1. The training data discussed in Section 3.2 will be used to fit the best-line latency-distance mapping. Essentially, CBG involves pinging the target from many monitors, determining the maximum possible distance from each monitor, creating a set of disks on Earth with the radii of each of these distances, and then intersecting all of these disks.

The first part of this stage is to check a cache for the query. If a cache entry for the /24 that the IP is in exists, this cache entry is returned. If not, ICMP `ping` measurements are performed to the target, or TCP `ping` measurements if the target isn't responsive to ICMP pinging. These measurements will be performed by a small number of servers located in different areas of the world. The measurements are collated, and are then used to perform Constraint-Based Geolocation. Training data (taken from the ground-truth data described in section 3.2) will be used to find the optimal "best-line" for mapping latencies to distances, as described in the Constraint-Based Geolocation paper.

This stage will be considered to be *conclusive* (for an IP address with a given WHOIS country code) when the following conditions hold:

- The polygon returned by Constraint-Based Geolocation does not include the country.
- The only country intersecting with the polygon returned by Constraint-Based Geolocation is the advertised country.

3.6 RIPE Atlas Stage

In this stage, measurements will be collected using the RIPE Atlas API. This API allows users to select probes and anchors from all over the world, and initiate latency measurements from them to targets of the user's choosing. As of writing, there are over 11,000 measurement servers in 172 different countries. Usage of the RIPE Atlas API is rationed through a credit system. This,

Algorithm 1: Constraint-Based Geolocation [36]

```
Pre:    best_line_distance maps latencies to distances, using the line fitted to training data
        disk_on_globe(position, radius) returns a disk on the globe with centre position
        and radius radius
Input:  $\mathbf{m}$, such that $\mathbf{m}_i$ is the position of the $i$'th monitor
        $\mathbf{p}$, such that $\mathbf{p}_i$ is the ping latency from monitor $i$ to the target
Output: Polygon in which the target could lie.
1 Disks ← [ ];
2 for $i \leftarrow 0$ to $|\mathbf{m}|$ do
3     Distance = best_line_distance($\mathbf{p}_i$);
4     Disks.append(disk_on_globe($\mathbf{m}_i$, Distance));
5 end
6 Intersection ← $\bigcap_{i=0}^{|\mathbf{m}|}$ Disks;
7 return Intersection
```

Algorithm 2: Posit [40]

```
Input:  $\mathbf{m}$, such that $\mathbf{m}_i$ is the position of the $i$'th monitor
        $\mathbf{l}$, such that $\mathbf{l}_i$ is the position of the $i$'th landmark
        $\mathbf{p}^m$, such that $\mathbf{p}_i^m$ is the ping latency from monitor $i$ to the target
        $\mathbf{p}^l$, such that $\mathbf{p}_i^l$ is the set of ping latencies to landmark $i$
Pre:    Compute $\hat{p}_m(d \mid l)$, an estimator giving the likelihood of the target being distance
        $d$ away from the monitor, given that the monitor records a latency of $l$ to that
        target. Implemented by training a KDE using $\mathbf{p}^l$.
        Compute $\hat{p}_l(d \mid v)$, an estimator giving the likelihood of the target being distance $d$
        away from the landmark, given an L1-threshold distance of $v$ between the target
        and the landmark, using training targets.
Output: Most likely location of the target
1 Function Likelihood($x$, $\mathbf{v}$)
2     MonitorScore ← $\sum_{i=0}^{|\mathbf{m}|} \log \hat{p}_m(d(x, \mathbf{m}_i) \mid l_i)$;
3     LandmarkScore ← $\sum_{i=0}^{|\mathbf{l}|} \log \hat{p}_l(d(x, \mathbf{l}_i) \mid \mathbf{v}_i)$;
4     return MonitorScore + LandmarkScore
5 end
6 $\mathbf{v}$ ←{l1_threshold_distance($\mathbf{l}_i, \mathbf{p}^m$) | $\mathbf{l}_i \in \mathbf{l}$}
7 $\mathbf{C}$ ← Constraint-Based-Geolocation($\mathbf{m}$, $\mathbf{p}^m$);
8 $\mathbf{C}_\mathbf{l}$ ← $\{m \in \mathbf{m} \mid \mathbf{C}$ contains $m\} \cup \{l \in \mathbf{l} \mid \mathbf{C}$ contains $l\}$;
9 return $\text{argmax}_{x \in \mathbf{C}_\mathbf{l}}$ Likelihood($x$)
```

combined with the fact that credits are relatively hard to earn/obtain, means that minimising the number of credits used is a key aim of this stage.

Measurements will be collected using the RIPE Atlas API, since PlanetLab is in the process of being decommissioned. PlanetLab would otherwise be an attractive choice, since measurements would effectively be free, and there could be more control over the measurements made – in particular, `jping-api` could have been deployed directly to PlanetLab servers. However, using RIPE Atlas does have some advantages – they have a larger number of monitors, and they provide a clean API for selecting probes matching certain criteria.

In the RIPE Atlas network, `ping` measurements are half the price of `traceroutes`. As such, using a ping-only geolocation algorithm will save credits. Additionally, as has been discussed in the literature many times (including by Candela et al. [51]), the accuracy of a latency-based geolocation algorithm is heavily dependent on the number of monitors used in close proximity to the target.

These two observations lead us to the geolocation algorithm used in this stage. This stage will be using an adapted, "iterative" version of Constraint-Based Geolocation. The algorithm will begin with a small number of monitors chosen from within the feasible region found by the measurements performed in Stage 1 of the system. Selecting only monitors in this region means that the selected monitors will be closer to the target, and as such their latency measurements will be more useful. These measurements will be fed to the regular Constraint-Based Geolocation algorithm, which will return a feasible region. If the region is not conclusive with respect to the IP's advertised country, more monitors will be selected within the new feasible region, and these latency measurements will be added to the existing ones. Constraint-Based Geolocation will then be performed again with this extended set of latency measurements. This process will be repeated until the feasible region is conclusive, or until a certain number of iterations have been performed.

Algorithm 3: Iterative Constraint-Based Geolocation (adapted from [36])

```
Input:  feasible_region, the feasible region found by Stage 1 of the system
        country, the advertised country of the target
        target, the IP of the target
Output: Polygon in which the target could lie.
1  measurements ← [];
2  used_probes ← [];
3  iteration ← 0;
4  while not conclusive(feasible_region, country) or iteration == max_iteration do
5      region_probes ← select_probes_within_region(feasible_region);
6      probes ← region_probes \ used_probes;
7      measurements ← get_ripe_measurements(ip, probes);
8      feasible_region ← constraint_based_geolocation(measurements);
9      iteration ← iteration + 1;
10 end
11 return feasible_region
```

3.7 Caliper Stage

This stage will involve running the novel geolocation algorithm presented in this report, we have named "Caliper" (after the measurement tool). This stage will generate a "best guess" for the location of the target, to include in the returned result.

The algorithm uses a statistical geolocation approach (inspired in particular by Posit by Eriksson et. al [40]), but also makes use of network topology information from traceroute measurements. We have chosen to design a new geolocation algorithm, rather than use an existing one, because we weren't able to identify a well-known geolocation algorithm which had been developed specifically for use on a global scale. Caliper is built with this purpose specifically in mind.

Running Caliper requires the use of monitors, landmarks, and *training targets* (which will come from the training data of the system). Caliper will involve finding the most likely position of the target, given likelihood distributions for monitor-target distances given latencies, and for landmark-target distances given inferred latencies between landmarks and targets.

Caliper has been achieved by augmenting and improving on the ideas presented in [40]. In particular, both algorithms will involve scoring a list of candidate guesses, and returning the guess with highest score. Caliper's score function will be arrived at by augmenting Posit's:

$$\text{score}(x) = \sum_{i=0}^{|\mathbf{m}|} \log \hat{p}_m(d(x, \mathbf{m}_i) \mid l_i) + \sum_{i=0}^{|\mathbf{l}|} \log \hat{p}_l(d(x, \mathbf{l}_i) \mid \mathbf{v}_i)$$

We will also be using L1-threshold distance as the estimator used for landmark-target distance, as suggested in that paper. We also evaluated the simpler Canberra distance, but found that the correlation between the Canberra distance and physical distance is much lower.

Caliper will make the following improvements and augmentations to existing statistical-based approaches:

- Global solution search
- Likelihood weighting
- Landmark-target reliability weighting
- Use of population and geographical data
- Traceroute and intermediate router hostname analysis

3.7.1 Global Solution Search

In the existing statistical geolocation algorithms we have encountered in the literature, the candidate locations considered for the target are restricted to the positions of landmarks and monitors, within the "feasible region" found by Constraint-Based Geolocation. This means that the minimum possible error for a given target t will be the distance between t and the nearest monitor or landmark. This works well for the US, a small area in which it's possible to source many geographically distributed landmarks. However, on a global scale, it is harder to source geographically distributed landmarks, so the distance between any target and its nearest monitor or landmark would be considerably higher.

A solution to this is to attempt to perform a search over a wide range of possible candidate locations, rather than just those in which there is a monitor or landmark. A simple way of doing this would be to iterate over all points within the feasible region, and find the point with the highest likelihood. This approach will be referred to as a "grid search". The granularity of this grid search, i.e: how many points are considered within the feasible region, would need to be considered. A higher granularity would potentially lead to higher accuracy, but would be more computationally expensive.

3.7.2 Likelihood Weighting

In the original Posit algorithm, the "score" for a candidate location is computed by simply taking the sum of:

1. The sum of the (estimated) monitor-target likelihoods, given the observed latency measurements
    - These likelihood values are derived using the relationship between the latency between monitors and the target, and geographical distance.

2. The sum of the (estimated) landmark-target likelihoods, given the observed latency measurements
    - These likelihood values are derived using the relationship between the landmark-target L1-threshold distances (derived from the monitor-target and monitor-landmark latencies), and geographical distance.

However, it is not clear that the information from monitors and landmarks is equally useful. For example, the correlation between monitor-target latency and physical distance is stronger than that between landmark-target L1-threshold distance and physical distance. As such, we introduce a weighting system for likelihood contributions. With such a system, instead of computing:

$$\text{score}(x) = \sum_{i=0}^{|\mathbf{m}|} \log \hat{p}_m(d(x, \mathbf{m}_i) \mid l_i) + \sum_{i=0}^{|\mathbf{l}|} \log \hat{p}_l(d(x, \mathbf{l}_i) \mid \mathbf{v}_i)$$

we can instead compute:

$$\text{score}(x) = \lambda_m \sum_{i=0}^{|\mathbf{m}|} \log \hat{p}_m(d(x, \mathbf{m}_i) \mid l_i) + \lambda_l \sum_{i=0}^{|\mathbf{l}|} \log \hat{p}_l(d(x, \mathbf{l}_i) \mid \mathbf{v}_i)$$

where λ_m and λ_l are weighting coefficients for monitor-target likelihood values and landmark-target likelihood values respectively.

In order to derive optimal values of λ_m and λ_l, we can optimise for performance over a set of validation data.

3.7.3 Landmark-Target Reliability Weighting

Another aspect of this score function which can be improved is in how landmark-target likelihood data is used. L1-threshold distance is used as an estimator for the physical distance between a landmark and a target. L1 threshold distance is calculated in two stages, given latencies to the target l^t, and latencies to the landmark l^l. Firstly, we calculate indices I such that:

$$I = \{k : l^l_k < \lambda_{lat} \text{ or } l^t_k < \lambda_{lat}\}$$

The intuition behind this calculation is that we only want to use latencies which can be useful – if both the latency to the landmark and the latency to the target are high, this latency pair is not likely to be useful, since the correlation between latency and physical distance degrades significantly as latency increases [40].

After these indices are calculated, the L1-threshold distance is calculated as:

$$\frac{1}{|I|} ||l^l(I) - l^t(I)||_1$$

where the notation $l(I)$ denotes the vector containing elements of l at indices contained in the set I.

One issue with this approach is that, in some cases, the I set will be very small, because many of the latency measurements were above λ_{lat}. This is particularly the case when using a Posit-based approach on a global scale with relatively few monitors, since in lots of cases all latency measurements will be relatively high.

We propose that the more latencies used to estimate the landmark-target distance, the better that estimate will be. Thus, we add an additional weighting to each landmark's contribution to the overall score of a candidate location. The score computation will thus be augmented to:

$$\text{score}(x) = \lambda_m \sum_{i=0}^{|\mathbf{m}|} \log \hat{p}_m(d(x, \mathbf{m}_i) \mid l_i) + \lambda_l \sum_{i=0}^{|\mathbf{l}|} \frac{|I_i|}{|\mathbf{l}_i|} \log \hat{p}_l(d(x, \mathbf{l}_i) \mid \mathbf{v}_i)$$

The added $\frac{|I_i|}{\mathbf{l}_i}$ coefficient means that the more latencies are used in calculating $\mathbf{v}_i$ (the L1-threshold distance between the target and landmark i), the more the likelihood counts towards the final score of the candidate location.

3.7.4 Population & Geographical Data

Certain candidate locations are inherently more likely than others, independent of the latency measurements to the target. For example, it is much more likely for a web server to be located in London than it is for it to be located in the Sahara desert. It is also very unlikely for a target to be located in a body of water such as an ocean or a sea.

Internet hosts tend to be located in cities and large settlements, as these areas have better internet infrastructure. These cities and large settlements can be identified as areas with high *population density*. In order to enhance Caliper's ability to identify the likelihood of a candidate location being correct, its scoring function can be further adapted to:

$$\text{score}(x) = \left(\lambda_m \sum_{i=0}^{|\mathbf{m}|} \log \hat{p}_m(d(x, \mathbf{m}_i) \mid l_i)\right) + \left(\lambda_l \sum_{i=0}^{|\mathbf{l}|} \log \hat{p}_l(d(x, \mathbf{l}_i) \mid \mathbf{v}_i)\right) + \lambda_p \hat{p}_p(x)$$

where $\hat{p}_p(x)$ gives an estimate of the likelihood of x as a candidate location given population density information, and λ_p acts as a weight on the contribution of the likelihood estimates derived from population density information. We define $\hat{p}_p(x)$ by taking the population density in the area around x, and to then divide it by the population density of the area with maximum population density.

3.7.5 Intermediate Router Hostname Analysis

An example of traceroute output to one of *HideMyAss!*'s "North Korean" VPN servers can be seen in Listing 3.1.

```
$ traceroute -I -q1 5.62.61.64
traceroute to 5.62.61.64 (5.62.61.64), 64 hops max, 72 byte packets
 1  brightbox.ee (192.168.1.1)  4.405 ms
 2  172.16.12.230 (172.16.12.230)  8.345 ms
 3  *
 4  213.121.98.128 (213.121.98.128)  13.654 ms
 5  87.237.20.138 (87.237.20.138)  14.790 ms
 6  lag-107.ear3.london2.level3.net (212.187.166.149)  15.587 ms
 7  *
 8  avast-softw.bear1.prague1.level3.net (212.162.8.142)  45.415 ms
 9  *
10  r-64-61-62-5.ff.avast.com (5.62.61.64)  47.289 ms
```

Listing 3.1: Traceroute to kp.hma.rocks (5.62.61.64) using ICMP pings and 1 packet per TTL.

It's clear from the path of the packets that the VPN server is not actually located in North Korea. One of the strongest indications of this is that the penultimate hop, `avast-softw.bear1-.prague1.level3.net`, contains a *location hint* indicating that it is located in Prague, and responds with a latency very close to that of the VPN server. This suggests that the VPN server itself is located close to Prague (which is the location of *HideMyAss!*'s headquarters).

Hint Type	Example Hostname	Example Location Hint
City Name	avast-softw.bear1.**prague**1.level3.net	Prague, CZ
City Name + Country Code	er1-ge-7-1.**londonuk**5.savvis.net	London, UK
City Name Abbreviation	cpe-68-173-83-248.**nyc**.res.rr.com	New York City, US
City Name Contraction	**lon**-tel-01gw.voxility.net	London, UK
City Name (Vowels Removed)	static-50-47-60-130.**sttl**.wa.frontiernet.net	Seattle, US
Airport Code (IATA)	ge-2-1-0.mpr1.**lhr**2.uk.above.net	London, UK
UN/LOCODE	16.151.88.129.**krsel**19d.kor.hp.com	Seoul, KR

Table 3.3: Examples of hostname location hint types. Certain hostnames taken from [29].

If it is possible to extract the city name from a hostname, it then becomes possible to use them to create constraints on the locations of further hosts in the path of the packets. For example, in the listing discussed above, we could estimate that the VPN server is within $47.289 - 45.415 = 1.874$ms of Prague.

Backbone internet providers (Tier 1 and 2 ISPs in particular) regularly add these kinds of location hints to their hostnames, presumably in order to be able to see at-a-glance where routers are located (making the jobs of network engineers easier). The exact format of location hints does not conform to any particular standard, but Table 3.3 shows a list of common types of hints.

A naive approach to location hint extraction would be to simply compile a list of possible location hints, and then to trust these hints without any verification. To demonstrate the issue with this approach, consider the hostname “cpe-60-177-11-428.eth.rr.com”. This hostname contains a “eth” component, which is the IATA code of Eilat Airport in Israel. However, this host is in fact located in the USA, and the “eth” hostname component is simply an abbreviation for “ethernet”.

A better approach, and the one which will be used in Caliper, is to verify these hostname hints. One way to perform this verification is to perform a WHOIS lookup on the host in question, and check that the hint city falls within that country. Backbone providers and large ISPs tend to use the correct WHOIS country code for their routers, so we believe trusting this data is safe.

For the purposes of this hostname location hint scraping process, the term *components* will describe the hostname split on non-alphanumeric characters. For example, **lon**-tel-01gw.voxility.net would be split into the components of **lon**, tel, 01gw, voxility and net. Often backbone providers will have multiple routers in one location, so they will append numbers within these location hint components - such as in ge-2-1-0.mpr1.**lhr2**.uk.above.net. Numbers never contain useful location information, so they can simply be filtered out. This final, filtered hostname component can then be looked up in the precompiled list of potential hostname hints. If found and successfully verified, the co-ordinates of this city can also be determined, and used to create an extra constraint on the position of the target.

We can estimate the latency between a hop and our target by subtracting the latency measured to the target from the latency measured to the hop, as described in [43].

Once we have our location hints, and our estimated hop-target latencies, we can use this as an extra way to contribute to the scores of candidate locations as follows:

$$\text{score}(x) = \ldots + \lambda_h \sum_{i=0}^{|\mathbf{h}|} \log \hat{p}_m(d(x, \mathbf{h}_i) \mid \hat{l}(x, \mathbf{h}_i))$$

where $\mathbf{h}$ is the set of hop locations identified in the traceroute to x (such that $\mathbf{h}_i$ is the position of the i'th hop), and $\hat{l}(x, \mathbf{h}_i)$ is the estimated latency between x and $\mathbf{h}_i$. λ_h acts as a weighting coefficient for likelihood contributions of intermediate hop constraints. This will be selected along with the other weighting coefficients using validation data. Note that the same likelihood distribution used for monitor-target latencies is being used here. As such, intermediate hops are essentially treated as “extra monitors” using estimated (instead of actual) latencies to the target. For our purposes, $\hat{l}(x, \mathbf{h}_i)$ will be defined simply as the latency to x, minus the latency to $\mathbf{h_i}$.

3.8 Pipeline Results

At the end of running the pipeline on an IP, a number of pieces of information will have been collected. This information will be the *result* of the pipeline for that IP.

- **IP Address** – the IP address the pipeline was run on.
- **Netblock** – the netblock that the IP is in.
- **WHOIS Country** - the country code of the netblock the IP is in.
- **Result** – the result of running the active geolocation stages of the pipeline. This can have multiple values:
  - **CORRECT** – the advertised country of the IP is demonstrably correct.
  - **INCORRECT** – the advertised country of the IP is demonstrably incorrect.
  - **INCONCLUSIVE** – it wasn't possible to conclude whether the advertised country of the IP is demonstrably correct or incorrect.
  - **NON_RESPONSIVE** – the IP didn't respond to ping measurements.
  - **ANYCAST** – the IP is anycast, meaning servers in multiple geographical locations are using that IP.
- **Parent Netblock** – information about the parent netblock of the netblock the IP is in.
- **Penultimate Hop** – information about the penultimate hop determined when tracerouting the IP.
- **Feasible Region** – the region that the IP is definitely within, determined by the CBG stages.
- **Best Guess Location** – the best guess for the location of the IP, determined by Caliper.

3.9 Web Interface

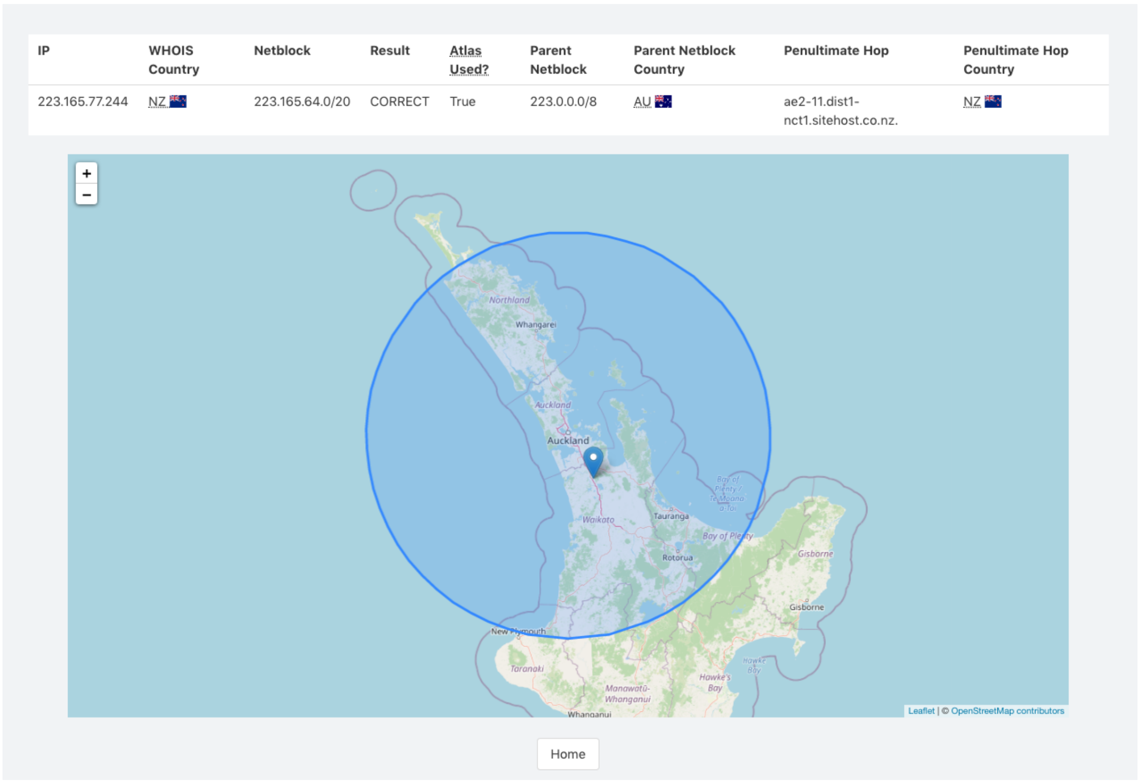
A web interface for the system will be created, in order to expose the functionality of the system to end users. This web interface will allow users to enter IP addresses to look up, and will then let them view the results in a readable way. The web interface designed is suitable for interested parties – for example law enforcement agencies or internet security firms – who need to look up individual IPs on demand.

Figure 3.3 shows an example of the results page for an IP address located in Auckland, New Zealand. As can be seen, the first section of the page displays a number of pieces of information in a table. These pieces of information are as described in Section 3.8. The "Penultimate Hop" column includes the hop's hostname, or the IP if the hop has no reverse DNS pointer. Flags are displayed next to country codes, in order to make output as easily-readable as possible. The "Atlas Used" column indicates whether the IP went through the RIPE Atlas stage or not. Underneath this table is a map which displays the feasible region – the region that the IP must be inside. The best guess, as determined by Caliper, is displayed using a marker.

3.10 Batch Processing

In addition to having a web interface for users to submit individual IPs for geolocation lookups, the system will also be able to accept batches of IP addresses for processing. The batch processing functionality will be used to perform analysis on the prevalence of geolocation fraud among large sets of IP addresses.

Instead of returning results in the form of a web interface, results will be outputted in a format suitable for loading into a spreadsheet, or manipulating with command-line tools (`awk`, `cut`, etc). In particular, output will be formatted as TSV (Tab-Separated Values) file. Having the output data in an easily manipulable format will make data analysis easier.



IP	WHOIS Country	Netblock	Result	Atlas Used?	Parent Netblock	Parent Netblock Country	Penultimate Hop	Penultimate Hop Country
223.165.77.244	NZ 🇳🇿	223.165.64.0/20	CORRECT	True	223.0.0.0/8	AU 🇦🇺	ae2-11.dist1-nct1.sitehost.co.nz.	NZ 🇳🇿

Figure 3.3: Screenshot of the design of the web interface geolocation results page. The page displays results for an Auckland-based server with correct geolocation.

3.10.1 Human Readable Region Descriptions

Displaying the information shown in the table in the web interface is straightforward in the batch processing TSV output. However, the restriction of TSV being text-only means that we must find a different, text-based way to record the feasible region for each IP.

One option would be to include the feasible region translated into a text format such as Well-Known Text (WKT). This would transform the polygon into a string such as `"POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10)"`. This is not a good strategy for a number of reasons. First, this string will increase in length as the polygon gets more complicated. This makes the TSV file harder to read. Second, this format is hard to understand – for the reader to understand this data (without using an external tool), they must be able to read the string of latitudes and longitudes, and visualise in their head where all of them are on the globe.

A better way to display this data is to give the feasible region a short, human-readable description. This description will let the reader immediately see which parts of the world are included in the polygon.

Our system will generate the polygon by building up the most specific string possible which sufficiently describes the polygon. This will be achieved by spitting the world up into its constituent parts:

- **Countries**: e.g: GB, FR, DE, PL...
- **Regions**: e.g: Northern Europe, South-Eastern Asia, Micronesia, ...
- **Continents**: e.g: Europe, Africa, Asia, ...

In order to find the most specific description for a polygon, each type of region will have a certain score:

- **Countries** will each score 1.

- **Regions** will each score 4.
- **Continents** will each score 9.

A description can be formed either by a list of countries (up to three of them) joined by a "/", or by any combination of two of the regions described above.

The score of a given description will be the sum of the score of each part of that description. For example, "GB/FR" will score 2, "Western Europe" will score 4, and "Europe" will score 9. The description with lowest score will be taken.

In order to find a set of candidate descriptions to evaluate and select from, a list of countries which intersect with the polygon is determined. If this list consists of 3 or less countries, these are concatenated together and returned. If not, a list of regions and continents which include these countries is compiled. Then, all combinations of up to two of these regions are considered and scored. The combination with lowest score is returned. The time complexity of this operation is $O(n^2)$ in the number of regions, as defined above, that the polygon spans.

3.11 Measurement Servers

In order to perform the Fast-Response stage, our system will need to perform measurements from a number of servers (controlled by us) located around the world. These monitors will need to be able to perform pings and traceroutes, and will also need to be able to expose this functionality through an API of some kind.

3.11.1 Ping and Traceroute Implementation Objectives

When designing our Fast-Response ping and traceroute software, we had a number of objectives in mind:

- **Ability to perform measurements to multiple machines at once (parallelism)**
  - This is desirable because it will speed up collecting large number of ping measurements, for example when collecting measurements in bulk to use for training.
- **Support for multiple ping types**
  - This is desirable because some targets will respond to one kind of ping, but not another. Therefore the more types of ping supported, the higher the chance that measurements to the target can be collected.
- **Ability to customise packet count, hop limit and timeout**
  - Using a higher packet count for ping and taking the minimum will mean the result more closely matches the "true" latency. Decreasing the hop limit when tracerouting can make tracerouting faster. Decreasing timeout can make pings and traceroutes faster.
- **IPv6 support**
  - The internet is slowly moving towards the usage of IPv6, so if we want our measurement software to remain useful in the future we should support this.
- **Tamper Resistance**
  - Many ping tools will include a timestamp in the ICMP packet sent, and rely on the same timestamp being sent back by the target in order to calculate latencies. An attacker can tamper with this, as described in [52]. The ping tool used in this project should not be vulnerable to these attacks.

Tool	Parallelism?	Multiple Ping Types?	Customisable?	IPv6?
ping	✗	✗	✓	✗
oping	✓	✗	✓	✓
fping	✓	✗	✓	✓
hping3	✗	✓	✓	✓
nping	✗†	✓	✓	✓

Table 3.4: Comparison of existing ping tools
† - nping reports supporting parallelism, but this feature appears to be broken.

Tool	Parallelism?	Multiple Ping Types?	Customisable?	IPv6?
tracepath	✗	✗	✗	✓
traceroute	✗	✗	✓	✓
tcptraceroute	✗	✓	✓	✓
nmap	✗	✓	✓	✓
mtr	✗	✓	✓	✓

Table 3.5: Comparison of existing traceroute tools

3.11.2 Existing Ping and Traceroute Tools

One option considered when planning the software to be deployed to the measurement servers was whether to simply use existing command-line ping and traceroute tools, and then parse their output.

Tables 3.4 and 3.5 show a comparison of the functionality of existing ping and traceroute tools. As the tables show, none of them fully meet the objectives previously mentioned.

3.11.3 jping and jping-api

Because none of the existing tools meet the objectives laid out in Subsection 3.11.1, a new tool (named `jping`) will be written. Developing this tool will involve writing code to perform pings and traceroutes. A HTTP API to allow this library to be called remotely (named `jping-api`) will also be written. `jping` and `jping-api` will support both ICMP and TCP pinging and tracerouting, to both IPv4 and IPv6 targets.

`jping` and `jping-api` will allow callers to customise their ping measurements - the number of packets sent, the timeout and the method (ICMP or TCP) will all be customisable. Traceroute measurements will be similarly customisable, with callers being able to specify the maximum hop limit, number of packets to send per TTL, and the traceroute method (ICMP or TCP).

Most importantly, `jping` and `jping-api` will be written with parallelism in mind from the bottom up. Both the `jping` library itself and the `jping-api` HTTP API exposing it will be able to process and deal with many requests in parallel.

`jping` will not use timestamps sent in their ICMP packets to calculate latencies, since this can be tampered with by attackers as described in [52].

Chapter 4

Implementation

In this chapter, details about the implementation of the design discussed in Chapter 3 will be described.

Two codebases have been used in this project. The first is `jping-api`, a Go library providing a HTTP API for performing pings and traceroutes. This library is discussed in Section 4.1. The rest of this chapter discusses the `ip_geolocation` Python library, which contains the bulk of the code for this project, including all of the geolocation algorithms used, code for querying the RIPE Atlas API and the code for the fraud detection pipeline itself.

4.1 jping and jping-api

In order to perform stages 1 and 2 of the system, `jping-api` will be deployed to a number of monitors located around the world. Netcraft have provided access to 8 DigitalOcean servers to use for these purposes. The full list of servers, their data centres and their approximate locations can be found in Table 4.1.

Hostname	City	Country	Data Centre	Latitude	Longitude
ip-geolocation-lon1.netcraft.com	London	UK	Equinix LD5	51.522629	-0.628996
ip-geolocation-nyc1.netcraft.com	New York City	US	Equinix NY7	40.797043	-74.031097
ip-geolocation-ams2.netcraft.com	Amsterdam	NL	TelecityGroup NL	52.293596	4.94155
ip-geolocation-sfo1.netcraft.com	San Francisco	US	Digital Realty SF	37.72397	-122.398035
ip-geolocation-sgp1.netcraft.com	Singapore	SG	Equinix SG2	1.321586	103.695912
ip-geolocation-fra1.netcraft.com	Paris	FR	Interxion PAR	50.120021	8.735696
ip-geolocation-tor1.netcraft.com	Toronto	CA	Equinix TR2	43.65104	-79.36193
ip-geolocation-blr1.netcraft.com	Bangalore	IN	NTT Bangalore	12.980725	77.588359

Table 4.1: List of measurement servers (provided by Netcraft).

4.1.1 jping

`jping` is written in the Go programming language. Go was chosen for this tool because its goroutines are a convenient way of spinning up threads to send pings, and because the `gopacket` library has good support for crafting raw packets and packet sniffing.

Pinging with jping

The code for the pinging section of `jping` can be found in `pinger.go`. Users of the library can instantiate a `Pinger` object, and then call its methods to get the results of ping measurements.

`jping` supports running measurements in parallel, performing pings with multiple packet types (ICMP and TCP–SYN), customising measurements, and supports pinging over IPv6. `jping`'s ping functionality works as follows, using "sending threads" and a "listening thread":

1. The sending thread will send a packet to the target (either ICMP or TCP–SYN, depending on requested type)
2. The sending thread will create a channel for the response to this packet to be sent on.
3. The sending thread will add this channel to a hashmap of "active request channels" accessible by the listener. This map will be indexed by some information that will allow the listener to identify the channel using only data in a given packet:
   - For ICMPv4, the address, sequence number and identifier are used.
   - For ICMPv6, the address, sequence number and identifier are used.
   - For TCP, the address and sequence (or acknowledgement) number are used.
4. The listener thread will see this packet being sent, and will record the timestamp at which it was sent in a hashmap in memory.
5. The sending thread will then wait on a Go channel for the response for this packet
6. The listener thread will then listen out for responses to the sending thread's packet.
   - All potentially relevant packets will be sniffed and decoded.
   - If the data in a packet corresponds to one of the current active requests, an object representing the ping measurement performed by this packet will be sent on the channel for that packet.
   - This object will contain the latency observed for this pair of packets, by subtracting from the timestamp of the received ping reply the timestamp recorded earlier for the ping request.
7. The sending thread will then receive the response on the channel, and will return the relevant result.

The listener thread uses the `gopacket/pcap` library to sniff for packets. It filters the packets using a filter expressed in Berkeley Packet Filter syntax – in particular, it uses the filter:

```
icmp or icmp6 or (tcp and ((tcp-syn|tcp-ack|tcp-rst) != 0))
```

This means it only has to process packets which could potentially be relevant to ping requests, which is more efficient.

One interesting bug encountered during the development of `jping-api` was a case where the API would stop working every time one particular IP address was pinged. The root of this bug was that the server behind this IP address was misconfigured to send duplicate ICMP Echo response packets to ICMP Echo requests. This corner case had not been considered, and a deadlock occurred in our code whenever this happened.

Tracerouting with jping

The `jping` library also has support for performing traceroutes, with a variety of parameters. The traceroute code (found in `route_tracer.go`) calls the ping code detailed in Subsection 4.1.1 in order to send pings with varying TTL. This means that the traceroute tool can perform traceroutes using all of the options supported by the ping tool (ICMP, TCP, IPv4 and IPv6). TTLs are looped over, and a ping is sent for each TTL in a separate thread. Each thread adds its result to a results array.

The pinging threads are synchronized using a `waitgroup` – the main thread `Wait()`s on this `waitgroup` after all the pinging threads have been started, and pinging threads who get a response from the target will call `Done()` on the `waitgroup` to release the main thread. Then the main thread can truncate the results array and return it.

We found that latency measurements would be inflated if too many ICMP packets were sent at one time. In order to avoid flooding the network with a huge number of ICMP packets, we introduced some delays in between sending pings. Our code has a delay of 100ms between sending each ping, but also sends pings in “chunks” of 10 – for example, before sending the 11th ping, all of the first 10 pings must have returned or timed out.

4.1.2 jping-api

`jping-api` provides an API which allows the `jping` library to be interacted with remotely via HTTP. `jping-api` is also written in Go, which means that it can make calls to the `jping` library directly. This is a much cleaner interface than would be possible with existing ping tools. Existing ping tools give output as text from command line tools, which would then need to be parsed. This is undesirable since the output from these tools is liable to change between tool versions.

API Specification

`jping-api` will serve its API on two endpoints, `/ping` and `/traceroute`. `/ping` will accept POST requests containing JSON payloads with the following structure:

- `ips` - type `[string]`
  - List of IPs to perform measurements to
- `method` - type `[string]`
  - Either "tcp" or "icmp", depending upon which ping type is desired
- `count` - type `int` - optional
  - The number of ping packets to send to each IP, default 10
- `timeout` - type `int` - optional
  - The timeout to use before deciding an IP is non-responsive, in milliseconds, default 1000

If successful, `/ping` will return JSON objects with the following structure:

- `ips` - type `[ping_result]`
  - List of ping results, each with the following format:
  - `ping_result` - type `object`, with keys:
    - `"ip"` - type `string` - the IP that the ping was sent to
    - `"up"` - type `bool` - whether the IP responded to pings within the timeout period
    - `"min_rtt"` - type `int` - the minimum round-trip-time observed to the IP, in picoseconds

`/traceroute` will accept POST requests containing JSON payloads with the following structure:

- `ips` - type `[string]`
  - List of IPs to perform measurements to
- `count` - type `int` - optional
  - The number of packets to send per TTL, default 5

- `max_hop` - type `int` - optional
  - The maximum TTL to send, default 32

If successful, `/traceroute` will return JSON objects with the following structure:

- `ips` - type `[traceroute_result]`
  - List of traceroute results for each IP, where each result has the following format:
  - `traceroute_result` - type `object`, with keys:
    - `"ip"` - type `string` - the IP that the traceroute was sent to
    - `"hops"` - type `[traceroute_hop]`, where each `traceroute_hop` consists of an array of objects representing routers encountered at that TTL, each object having form:
      - `"ip"` - type `string` - the IP of the intermediate router, if present
      - `"hostname"` - type `string` - the hostname of the intermediate router, if present
      - `"timeout"` - type `bool` - whether the measurement for this intermediate router timed out
      - `"rtt"` - type `int` - the round-trip-time observed to this intermediate router, in picoseconds

4.1.3 Architecture & Deployment

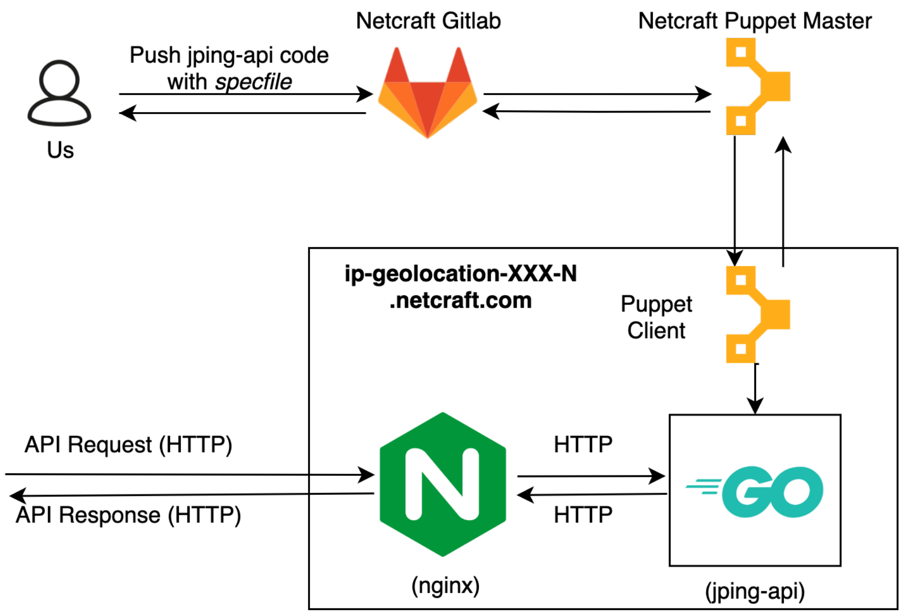


Figure 4.1: Architecture used to serve and deploy jping-api.

Figure 4.1 shows how `jping-api` is deployed and served. All 8 monitors are Digital Ocean droplets running CentOS 7, using this architecture. We will now describe each part of the diagram, starting in the top left and moving clockwise.

Pushing code to Gitlab & writing specfiles

In order to install `jping-api` on the CentOS 7 machines, we need to write a “specfile” – essentially a recipe for building an RPM package. RPM packages are used to facilitate software installation on a number of Linux distributions, including CentOS 7. Our RPM package will include a number of files, which will need to be placed in certain locations on the measurement servers:

- Nginx configuration files

- systemd configuration files
- The jping-api binary itself

The specfile is where we specify these files, and where they will be moved to when installing. Instructions for building the jping-api binary are also included in the specfile. The specfile additionally specifies that a new user (`jping-api`) will be made. This user will be the user used when running `jping-api`. Finally, the specfile will give the `jping-api` binary special capabilities, so that it can create and use raw sockets. The specfile can be found under the name `jping-api.spec` in the `jping-api` repository.

Puppet

Puppet is an open-source configuration management tool, which allows infrastructure to be specified in a code-like language. Using Puppet in this situation saves a great deal of time, since configuration files specifying the deployment of `jping-api` can be written once, and then deployed to all 8 monitors automatically. The Puppet configuration to deploy `jping-api` was not written by us – Netcraft have written this. This is the only Netcraft-written section of `jping-api`, and only affects how the package is deployed, and not any of the functionality of the package.

jping-api HTTP server & systemd

The `jping-api` binary will be moved onto the measurement server, as specified by the specfile used to build the `jping-api` RPM. However, we need some way of running this binary, such that it will be restarted if (for example) a monitor is restarted. `systemd` is an ideal tool for this. Among other things, `systemd` allows users to specify "services" which will be run when a machine is powered on. Services are specified using files with a `.service` extension – our one is named `jping-api.service` and can be found in the jping-api repository. Our service will simply execute the `jping-api` binary, getting it to listen for traffic on localhost on an unused port.

Nginx

In front of the Golang HTTP server, we are using Nginx to handle incoming requests from the outside world. This is generally considered to be good practice, and has a number of specific benefits for us:

- Nginx has excellent support for handling HTTPS connections
- If our Golang server ever goes down, Nginx will return meaningful HTTP errors. Without Nginx in front, the connections would simply time out with no response.

The Nginx server takes incoming requests on port 443 (the standard port for HTTPS connections), and relays them to our Golang HTTP server listening on an unused port on `localhost`. This is done using the `proxy_pass` directive. The Nginx configuration file used can be found in `jping-api.conf` in the `jping-api` repository.

4.2 Fraud Detection Pipeline

This section will discuss the code used for the fraud detection pipeline itself. The `ip_geolocation` Python package contains the code for this section of the project. The Python language was chosen for a number of reasons:

1. The Caliper algorithm involves statistical computing – in particular, the use of Kernel Density Estimation – and Python has excellent libraries for this.

2. Python has a number of excellent geo-data manipulation package packages (`shapely`, `geopy` and `pyproj`), all of which interact cleanly with one another.

3. Python bindings have been written for RIPE Atlas's API (the `ripe.atlas.cousteau` package).

The package consists of a number of subpackages which are relevant to the fraud detection pipeline:

```
ip_geolocation/
 │
 ├─ utils/
 │
 ├─ inetnum/
 │
 ├─ measurement_sources/
 │
 ├─ intermediate_hop/
 │
 ├─ geolocation_algorithms/
 │
 └─ fraud_detection/
```

The `utils` subpackage contains a small number of utility functions which are required in multiple different other subpackages – for example, a function to get a `shapely` disk with a given radius. The other packages will now be discussed in more detail.

4.2.1 Inetnum Database

The `ip_geolocation.inetnum` subpackage includes an `InetnumDatabase` class, objects of which are able to interact with Netcraft's inetnum database. This database contains IP netblocks and data on these netblocks – most importantly (for our purposes) their country code. This functionality could be replaced by performing WHOIS querying, but this involves network calls which are slower than local database queries.

4.2.2 Measurement Sources

The `ip_geolocation.measurement_sources` subpackage contains code for querying both the RIPE Atlas API and the measurement servers running `jping-api`. In both cases, `LatencyMeasurement` instances are returned. This abstraction means that measurements can be used interchangeably, and new measurement sources could be added easily in future.

The `jping-api` servers are queried by simply sending a HTTP POST request to each server, as specified in Subsection 4.1.2. This is done using a threadpool, so the measurements are collected in parallel. Querying the RIPE Atlas API is more complicated.

Querying the RIPE Atlas API

The `ripe.atlas.cousteau` package includes API bindings for initiating measurements to a target. The API call to create a measurement requires a list of probes. The `get_latency_measurements` function takes a region in which to select probes from. A list of probes (which is cached in memory and refreshed every 30 minutes) is then searched to find probes in the requested region. These probes are then passed to the call to the API which initiates the measurement. The function additionally has a parameter `iso_a2`, which takes a country code. If present, this will cause up to 10 probes from that country to be added to the probe list.

The function then waits for the results of these measurements to be ready. This is achieved by performing another API call every 5 seconds, until all of the results are ready, or it is determined that we should give up. Often probes won't respond, so waiting for all probes to respond is often futile. Determining when to give up is important – too early, and we miss measurements; too late and we waste time. An additional complication is that the RIPE Atlas API has a variable processing delay (we have seen anything from 10 seconds to 3 minutes). The solution we have decided upon is to wait for the first response, and then give up 15 seconds after this response. The logic behind this decision is that once the first response has been received, we will have already overcome the processing delay (which applies to all probes), and additional delays will be probe-specific.

An important limitation of the RIPE Atlas API is that the number of concurrent measurements is limited to 100. When running batches of IPs with many threads, we hit this limit regularly. To solve this problem, a `BoundedSemaphore` is used to synchronize access to the API. At most 8 threads can be making calls to the API at once. This has been sufficient to prevent hitting the concurrent measurement limit.

4.2.3 Intermediate Hop

The `ip_geolocation.intermediate_hop` subpackage includes code for extracting location hints from hops encountered when tracerouting targets. The `LocationHint` class defines an abstract representation of a location hint. The `iata_location_hints.py` file contains code for extracting and verifying hints from a hostname.

When the `iata_location_hints.py` is loaded, it will (if the list doesn't already exist on disk) generate a list of location hints using city names and airport codes. The following hints will be generated:

- IATA airport codes (e.g: LHR, ATL, etc.)
- IATA city codes (e.g: LON, NYC, etc.)
- City names (spaces removed) (e.g: LONDON, NEWYORK, etc.)
- City names (spaces and vowels removed) (e.g: LNDN, NWYRK, etc.)
- City names (spaces removed and country code added) (e.g: LONDONGB, NEWYORKUS, etc.)
- City names (spaces and vowels removed, and country code added) (e.g: LNDNGB, NWYRKUS, etc.)

In the region of 50,000 hints are generated in this way. These hints will then be used as described in the Design chapter (Subsection 3.7.5). Location hints will be validated by looking up the IP of the router via WHOIS and getting its country code. If the country code matches the country code of the hint, the hint is believed, if not it is discarded.

4.2.4 Geolocation Algorithms

The `ip_geolocation.geolocation_algorithms` subpackage contains Python implementations of three geolocation algorithms - Constraint-Based Geolocation, Posit and Caliper. Constraint-Based Geolocation and Caliper will be used in the pipeline.

Constraint-Based Geolocation

A `ConstraintBasedGeolocater` takes a list of `LatencyMeasurements` to the target and returns the region that the target could be within, according to the Constraint-Based Geolocation algorithm as presented in [36]. The code is a direct implementation of the approach described in that paper. The geolocater is trained by passing `LatencyMeasurements` to the `train` method. Scipy's `optimize` module is used to find the "bestline". An untrained `ConstraintBasedGeolocater` uses the (conservative) 2/3 speed of light baseline.

Caliper

The implementation of Caliper will be discussed in its own section of this chapter, Section 4.3.

Column	Type	Default Value
prefix	CIDR	
time_updated	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP
netcraft_region	BYTEA	
ripe_region	BYTEA	
penultimate_hop_ip	INET	
penultimate_hop_hostname	VARCHAR(255)	
penultimate_hop_timeout	BOOLEAN	
penultimate_hop_latency	INTERVAL	

Table 4.2: Schema for the prefix_cache table used for caching by the pipeline.

4.2.5 Fraud Detection Pipeline

This subpackage contains all the code used specifically by the Fraud Detection Pipeline. The entry point for the pipeline can be found in `ip_geolocation.fraud_detection.fraud_detection_pipeline`. A pipeline can be instantiated from the `FraudDetectionPipeline` class. The different stages are all in separate files in this module.

Each component of the `FraudDetectionPipeline` is designed with thread-safety in mind. This means it is possible to submit many IPs to one `FraudDetectionPipeline` object concurrently using multiple threads. Much of the work done by the pipeline is I/O bound, so submitting IPs concurrently can increase the throughput of the pipeline significantly.

Caching

As discussed in the Background section, except in the context of transit ASs, two IP addresses in the same /24 block will be in the same physical location. Thus, it is unnecessary to repeat measurements to IPs, if IPs with the same /24 prefix have already had measurements performed.

For this reason, FraudDetectionPipeline objects connect to a PostgreSQL database and cache measurement results for /24 prefixes. Results are cached in the `prefix_cache` table, the schema for which can be seen in Table 4.2.

PostgreSQL has support for IP address and IP network types (`INET` and `CIDR` respectively), which makes it an appropriate choice for this project. These types are used for the `prefix` and `penultimate_hop_ip` fields. The `time_updated` field exists so that the cache entry can be deleted after a given time. `netcraft_region` and `ripe_region`, fields which store the regions determined by the Fast-Response and RIPE CBG stages, have type `BYTEA`. This is because they store the WKB (Well Known Binary) representation of the regions.

Local Analysis Stage

In the local analysis stage, the first step is to determine the netblock for the IP address. This is done by querying an IP allocation database compiled by Netcraft. The same thing could be done using a WHOIS query instead, but this would introduce a delay due to the network access. The code to do this lookup can be found in `ip_geolocation.inetnum.inetnum`. Rows from the database are transformed into `InetnumNetblock` objects, which hold a number of pieces of data:

- **netblock_id** – the unique identifier of the netblock
- **parent_netblock_id** – the unique identifier of the parent of the netblock
- **network** – the network of the netblock itself, for example `192.168.0.0/16`
- **country** – the country code of the netblock
- **netname** – the name given on the inetnum object of the netblock

- **description** – the description given on the inetnum object for the netblock
- **source** – the source of the above information (usually one of the RIRs)

The country code of the netblock for the IP is used to compare against the actual location, when deciding whether the IP is using fraudulent geolocation. The parent netblock is easily determined, by using a `InetnumDatabase` object to lookup up a netblock with an id of the netblock's `parent_netblock_id`.

Finally, a traceroute is performed in order to get the penultimate hop for the IP address. This step assumes that `jping-api` is running on the machine. A HTTP request is made to the `/traceroute` endpoint, which returns data as described in Subsection 4.1.2. The IP from this query is looked up in the `InetnumDatabase` again to get information about the penultimate hop.

The results from this stage are saved, and will later be added to a `FraudDetectionResult`, which will be returned at the end of the pipeline.

Fast-Response CBG Stage

In this stage, measurements are collected from the servers in our control, using the code in `ip_geolocation.measurement_sources`. These measurements are then passed into a `ConstraintBasedGeolocater` in order to find the region that the IP could be located within. This `ConstraintBasedGeolocater` is trained using the data discussed in Subsection 3.2.

RIPE Atlas (Iterative CBG) Stage

This stage involves running "iterative CBG" using measurements from the RIPE Atlas API. The API is queried using the code in `ip_geolocation.measurement_sources`, as described in Subsection 4.2.2. Again, the `ConstraintBasedGeolocater` is trained using the training data described in Subsection 3.2.

The iterative CBG algorithm's initial region for selecting monitors will be derived from the previous stage's feasible region. This is important, because monitors in this region are likely to be more useful in geolocating the target, since they are likely to be closer to it.

The RIPE Atlas API has an endpoint for selecting probes within a certain region. Unfortunately, making this call is very slow when many probes are being returned. For example, if the feasible region determined by the previous stage spans an entire continent, a list of every single RIPE Atlas probe in that continent will be returned. The RIPE Atlas API paginates responses, so many requests would be needed to fetch such a list. The solution we have come to is to fetch and cache in memory the entire list of RIPE Atlas probes every 30 minutes. Probes aren't likely to change very often, and even if they did, our code is resilient to non-responsive probes, so 30 minutes seemed like a reasonable cache refresh interval. This refresh is done by a `threading.Timer` thread. Synchronisation is necessary in order to stop this thread and threads selecting probes from conflicting; a `threading.Lock` is used for this.

Caliper Stage

The Caliper stage will use measurements from previous stages to determine a "best guess" at the location of the target. The implementation of Caliper will be discussed in more detail in Section 4.3.

4.3 Caliper

The code for Caliper can be found in `ip_geolocation.geolocation_algorithms.caliper`. This file contains the definition of a `CaliperGeolocater`. The constructor of this class takes in three hyperparameters – `monitor_weight`, equivalent to λ_m, `population_weight`, equivalent to λ_p, and `hop_weight`, equivalent to λ_h. λ_l will be taken as 1.

4.3.1 Kernel Density Estimation

In order to learn $\hat{p}_m$ and $\hat{p}_l$, multivariate kernel density estimation (herein KDE) is used. In particular, the `train` method takes a list of measurements to landmarks, and a list of measurements to "training targets". These measurements are made to targets with known position. The landmark latencies are used to train the monitor-target KDE, and the training target latencies are used to train the landmark-target KDE. A multivariate KDE is fitted to pairs of latencies and their true geographical distances (from the passed landmark latencies) in order to get the monitor-target KDE. A multivariate KDE is fitted to pairs of L1-threshold distances and their true geographical distances (calculating L1-threshold distances using the passed training target and landmark latencies), in order to get the landmark-target KDE.

The particular KDE used is a `scipy.stats.gaussian_kde`, and Silverman bandwidth estimation is used. Plots of the KDEs can be found in Figure 4.2. The R^2 coefficient of determination has been calculated for each of the sets of data used for training. For the relationship between monitor-target latency and distance, we have found $R^2 = 0.6496$, which indicates a relatively strong correlation. The correlation between landmark-target L1-threshold distance and landmark-target physical distance is weaker, with $R^2 = 0.4235$. However, information derived using this relationship is still useful.

Optimising KDE Calls

We found that making calls to the KDE was a bottleneck in the speed of our code, so we have written a wrapper for `scipy.stats.gaussian_kde`, called `MemoizedBivariateKDE`, which evaluates the KDE over a grid of values and stores the results in a 2D array. Then, when a result from the KDE is needed, it can be returned from this precomputed array.

4.3.2 Parallelization

Caliper makes use of a `multiprocessing` process pool in order to parallelize its search for an optimal solution. The current implementation performs a grid search with a granularity of 1 degree. Constraint-Based Geolocation on the passed latency measurements is used to restrict the region over which optimisation is performed.

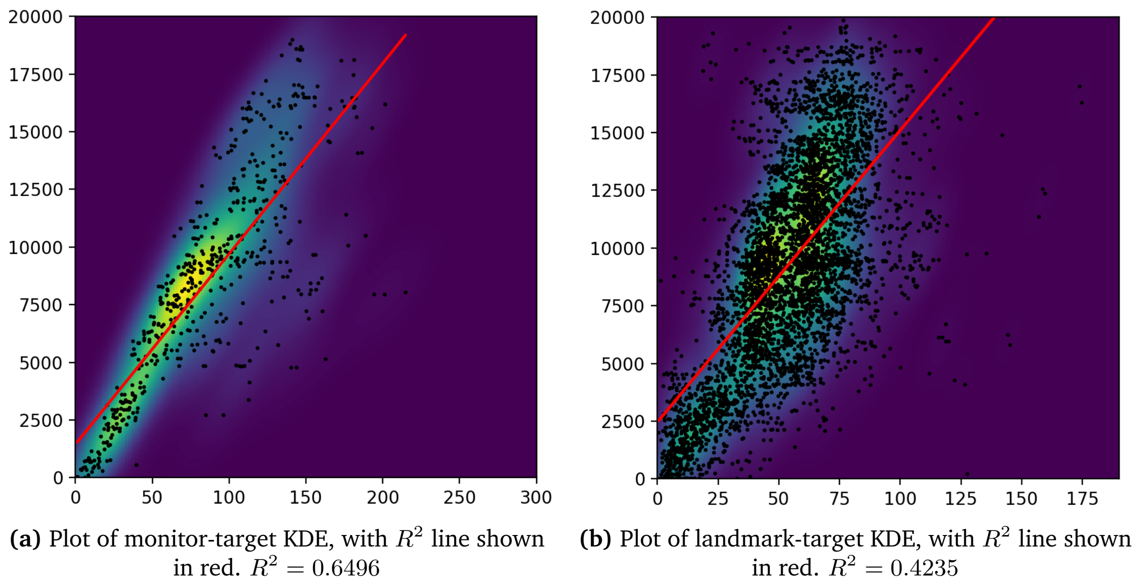
4.3.3 Hyperparameter Search

In order to find optimal values for Caliper's hyperparameters, a grid search over possible parameter values was performed. The performance of different values of λ_m and λ_p were evaluated on a set of 90 validation targets. These validation targets were separate from the training and testing targets in line with machine learning best practices. First, a coarse grained search was performed. The results of this search can be found in Figure 4.3. An optimum region was identified during this search, and then a finer-grained grid search was performed on that region, the results of which can be seen in Figure 4.4. The optimal hyperparameters found were $\lambda_m = 14$, $\lambda_p = 4$.

4.4 Web Interface

We chose Flask to implement the web interface for our geolocation system because it's straightforward and lightweight, and is implemented in Python, so interaction with our Python-based fraud detection pipeline is trivial.

For the same reasons as stated in the discussion about `jping-api`'s architecture, Nginx is being used. Nginx communicates with uWSGI, a micro-server which supports WSGI (Web Server Gateway Interface) – a Python-specific protocol for interaction between web servers and applications. uWSGI forwards requests on to the `ip_geolocation.front_end.server.app` Flask application.



(a) Plot of monitor-target KDE, with R^2 line shown in red. $R^2 = 0.6496$

(b) Plot of landmark-target KDE, with R^2 line shown in red. $R^2 = 0.4235$

Figure 4.2: Plots of monitor-target and landmark-target KDEs. Each black dot is one training instance. Lighter colours indicate that the KDE scores points in that region higher. Red lines are determined by linear regression, in order to find the R^2 value.

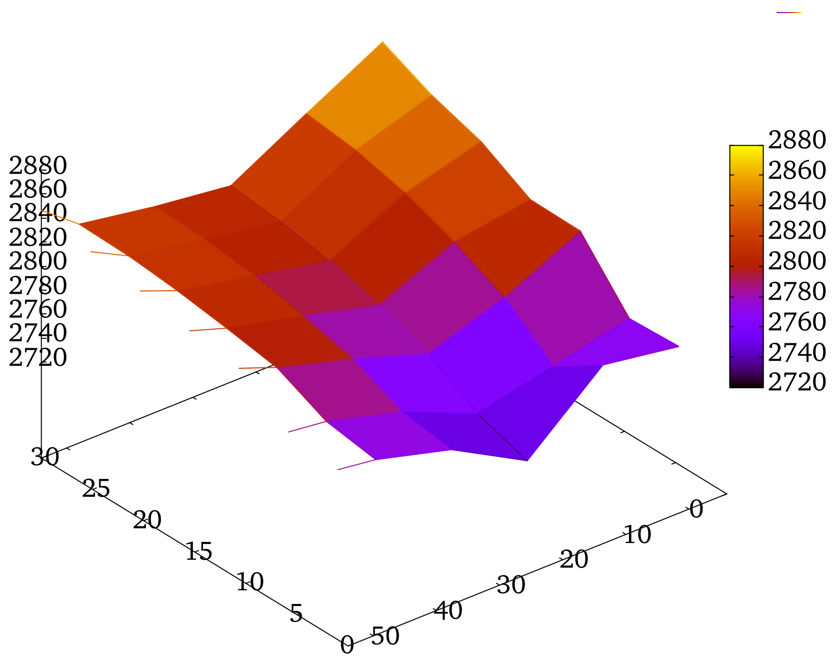


Figure 4.3: 3D plot showing the mean error achieved when varying Caliper hyperparameters using a coarse-grained grid search. λ_m has been varied from 0 to 30, and λ_p from 0 to 60. Values with $\lambda_m = 0$ are omitted due to them being much higher than all other values.

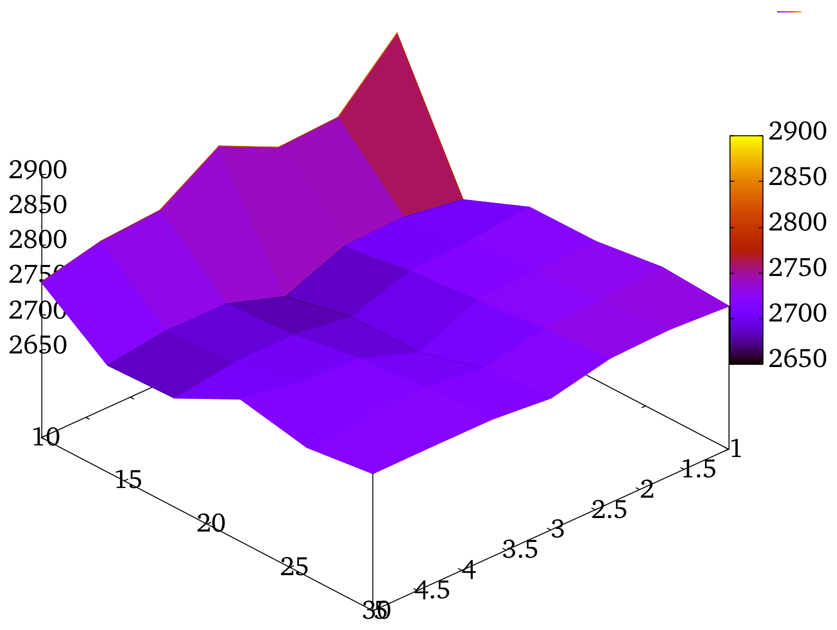


Figure 4.4: 3D plot showing the mean error achieved when varying Caliper hyperparameters using a fine-grained grid search. λ_m has been varied from 10 to 30. λ_p has been varied from 1 to 5.

Because running an IP through the IP geolocation pipeline takes a long time, an asynchronous task queue (Celery) is used. When a user requests that an IP is geolocated, the request is passed to one of the Celery workers through a Redis message broker.

Both the Flask application and the workers interact with a PostgreSQL database. Workers write their results to the `result` table, and the Flask application reads from this when a user requests a result.

If the user requests a result page for an IP which is currently being processed, they will be served a static waiting page. For convenience, this waiting page will auto-refresh every 30 seconds using a `<meta http-equiv="refresh" ...>` HTML tag. When a user submits a request to geolocate an IP, a POST request is issued by the JavaScript on the main application page. The response to this POST request will be a link to the result/waiting page for that IP address. The JavaScript will then forward the user's browser to this result/waiting page.

A tool called Flower, shown in Figure 4.6, is used to monitor Celery workers. The tool provides a web interface for viewing tasks and worker states. Load statistics and time elapsed per task can be viewed through this interface. Additionally, if an unhandled exception is thrown during a task running, details will be accessible through the Flower web interface.

4.5 Batch Processing

The `batch_process.py` script in the root of the `ip_geolocation` repository can be called with two arguments – a list of newline-separated IP addresses to process, and an output file path. The output file is read before processing, so that a list of IP addresses not already processed can be generated. This means that the batch processing can be stopped and resumed simply by stopping and restarting the script.

The script also takes a `-threads` argument, which lets the caller select a number of threads to use when processing IPs in the file. For the analysis we performed in Chapter 5, we used 8 threads. This is the default value of the argument.

Each time a thread gets a result for an IP, it appends its result to the output file in the format specified in Section 3.10. Examples of output from the batch processing feature of our system can be seen in the `report-data` directory of the code archive submitted alongside this report.

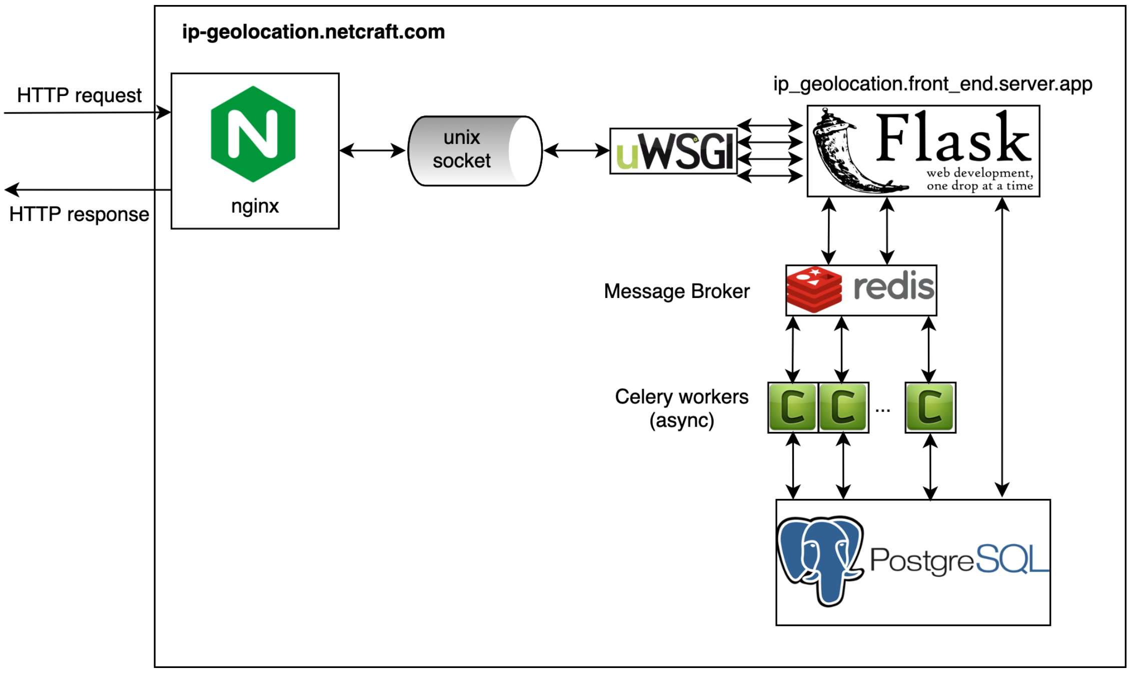
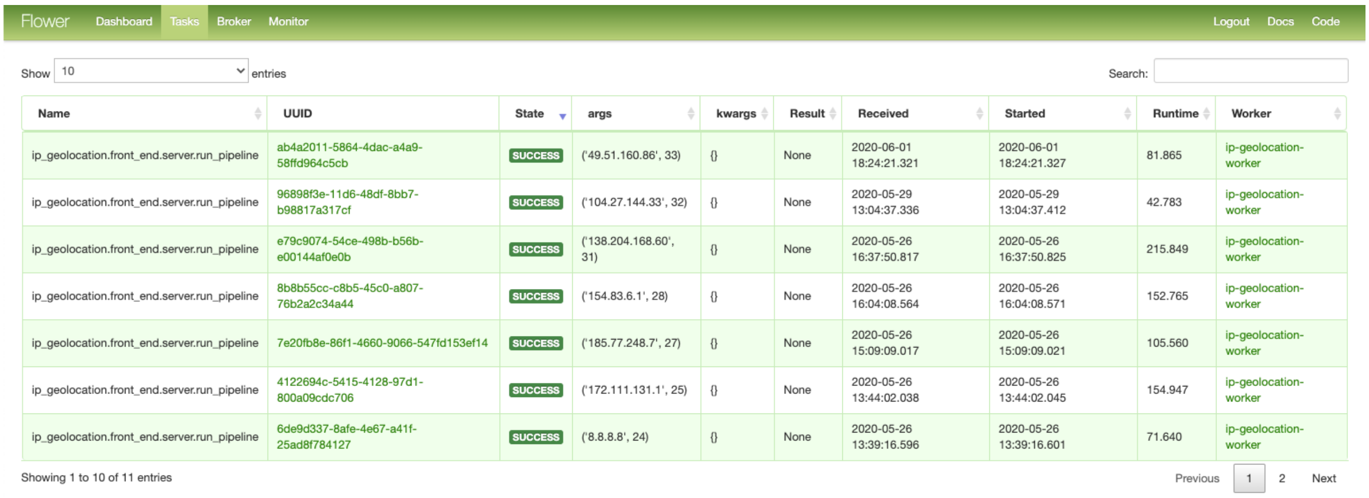


Figure 4.5: Architecture used on the web server serving the web interface.



Name	UUID	State	args	kwargs	Result	Received	Started	Runtime	Worker
ip_geolocation.front_end.server.run_pipeline	ab4a2011-5864-4dac-a4a9-58ffd964c5cb	SUCCESS	('49.51.160.86', 33)	{}	None	2020-06-01 18:24:21.321	2020-06-01 18:24:21.327	81.865	ip-geolocation-worker
ip_geolocation.front_end.server.run_pipeline	96898f3e-11d6-48df-8bb7-b98817a317cf	SUCCESS	('104.27.144.33', 32)	{}	None	2020-05-29 13:04:37.336	2020-05-29 13:04:37.412	42.783	ip-geolocation-worker
ip_geolocation.front_end.server.run_pipeline	e79c9074-54ce-498b-b56b-e00144af0e0b	SUCCESS	('138.204.168.60', 31)	{}	None	2020-05-26 16:37:50.817	2020-05-26 16:37:50.825	215.849	ip-geolocation-worker
ip_geolocation.front_end.server.run_pipeline	8b8b55cc-c8b5-45c0-a807-76b2a2c34a44	SUCCESS	('154.83.6.1', 28)	{}	None	2020-05-26 16:04:08.564	2020-05-26 16:04:08.571	152.765	ip-geolocation-worker
ip_geolocation.front_end.server.run_pipeline	7e20fb8e-86f1-4660-9066-547fd153ef14	SUCCESS	('185.77.248.7', 27)	{}	None	2020-05-26 15:09:09.017	2020-05-26 15:09:09.021	105.560	ip-geolocation-worker
ip_geolocation.front_end.server.run_pipeline	4122694c-5415-4128-97d1-800a09cdc706	SUCCESS	('172.111.131.1', 25)	{}	None	2020-05-26 13:44:02.038	2020-05-26 13:44:02.045	154.947	ip-geolocation-worker
ip_geolocation.front_end.server.run_pipeline	6de9d337-8afe-4e67-a41f-25ad8f784127	SUCCESS	('8.8.8.8', 24)	{}	None	2020-05-26 13:39:16.596	2020-05-26 13:39:16.601	71.640	ip-geolocation-worker

Showing 1 to 10 of 11 entries

Figure 4.6: Monitoring Celery workers with Flower.

Chapter 5

Experimental Results

The system described in this report has been used to analyse a number of sets of IP addresses. These IPs were processed using the batch processing feature of the system.

Analysis of these results is presented here to demonstrate that the system described in this report is capable of detecting geolocation fraud on a large scale. To the best of our knowledge, analysis of this kind has not been described in the literature before. In total, we have examined over 111,000 unique IP addresses, and have cached results for over 16,000 /24 blocks – address space spanning over 4 million IPs.

All in all, we found 62,518 individual servers using incorrect geolocation, and 225 out of the 249 possible country codes being used incorrectly. In total, 8,507 /24 blocks have been found to be using incorrect geolocation, which is address space spanning greater than 2.1 million IPs. The raw data used for all of this analysis (as generated by the batch processing feature of the system) can be found in the `report-data` directory of the submitted code archive.

5.1 Web Server Results

5.1.1 Tax Havens

The first dataset analysed is a set of web-server IP addresses which are in netblocks reporting to be in countries considered to be "tax havens". We analysed all IPs currently used by servers serving web content in netblocks marked as being in the Cayman Islands, the Seychelles, the Virgin Islands, or Dominica. This set of IPs have been taken from Netcraft's Web Server Survey, a comprehensive list of servers serving web content. All of these countries are small, have limited network capacity, and are considered to be tax havens.

Table 5.1 and Figure 5.2 show the results of this analysis. Of the web servers in these countries which responded to ping measurements, 74.7% of them were using demonstrably incorrect geolocation. The country with the highest rate of demonstrably incorrect geolocation was the Virgin Islands, at 98%.

Two of the largest blocks of IPs with incorrect geolocation are 165.231.0.0/16 and 196.196.0.0/16. Together, these blocks span 131,072 IP addresses, and both of these netblocks have Seychelles country codes. Every IP serving web content in these ranges is using incorrect geolocation. They are both owned by a Seychelles-based organisation called FiberGrid. The addresses appear to in fact be located in Sweden, as can be seen in the geolocation result seen in Figure 5.1. `4shared.com`, a popular online file sharing service, is using IPs in these netblocks.

5.1.2 Unlikely Country Codes

The next dataset examined was a smaller list of 515 IPs, each of which is in a netblock using an "unlikely" country code. The most surprising finding in this dataset was the number of IP addresses

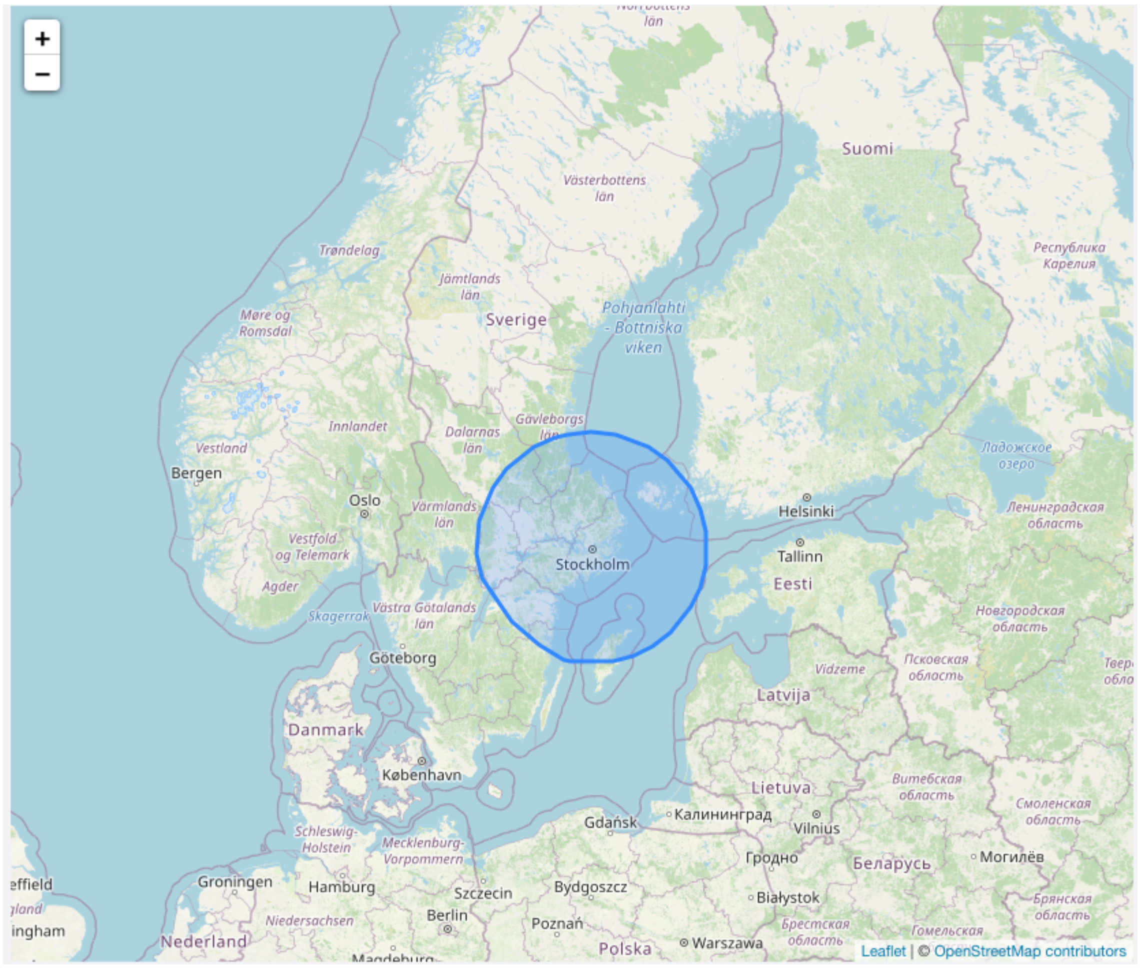
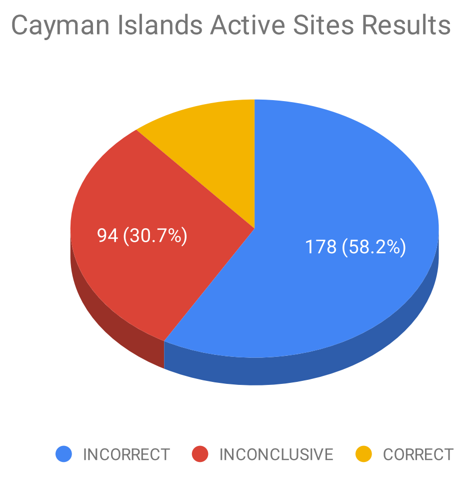


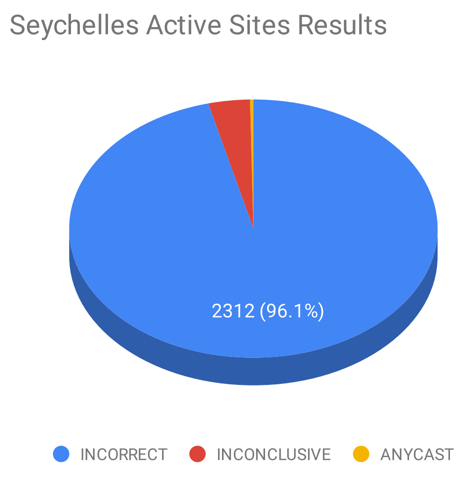
Figure 5.1: Result of iterative CBG using RIPE Atlas API measurements on an IP in a FiberGrid netblock (165.231.147.103) using Seychelles country code.

	Web Server IPs		**Result**			
Country Code	**Total**	**Responsive**	**Incorrect**	**Inconclusive**	**Anycast**	**Correct**
KY (Cayman Islands)	387	306	178	94	0	34
SC (Seychelles)	3491	2407	2312	88	7	0
VG (Virgin Islands)	1286	1154	1132	22	0	0
DM (Dominica)	1073	1068	1045	23	0	0
Total	**6245**	**4935**	**4667**	**227**	**7**	**34**

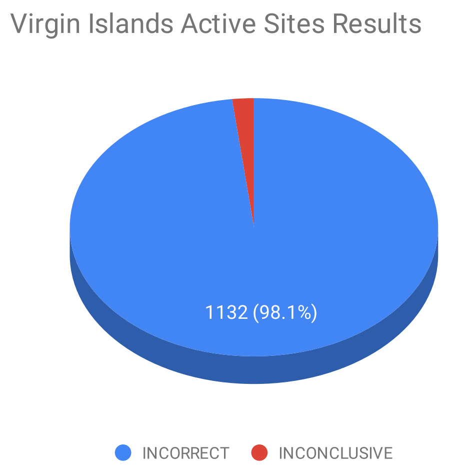
Table 5.1: Results of analysing IPs in tax haven netblocks.



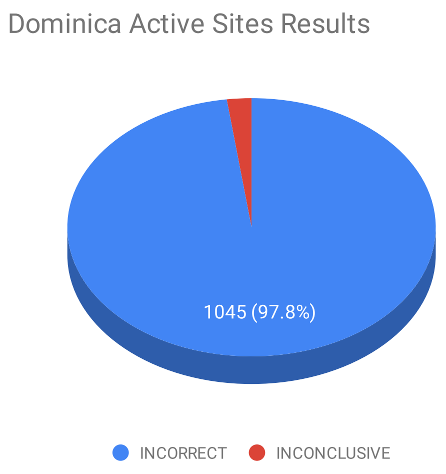
(a) Results from Cayman Islands IPs



(b) Results from Seychelles IPs



(c) Results from Virgin Islands IPs



(d) Results from Dominican IPs

Figure 5.2: Results of analysing ping-responsive IPs hosting web content in tax haven netblocks, as discussed in Subsection 5.1.1.

registered to Bouvet Island. Bouvet Island is the most remote island in the world, inhabited only by penguins and other sea birds. Bouvet Island isn't a sovereign state, but it has been allocated an ISO 3166-1 alpha-2 code [53], so it is possible to use it as the country code on an inetnum object. Of the IPs that responded to ping measurements, that were reportedly on Bouvet Island, 186 (98.9%) were using demonstrably incorrect geolocation. The other 1.1% were marked as being anycast IPs, which also means they aren't likely to be hosted on Bouvet Island. All of these IPs belong to one VPS hosting company, Njalla, which is run by Peter Sunde (the founder of Pirate Bay). These IPs have been used to host spam sites, such as `insuredlife.xyz` (fraudulently impersonating a UK insurance firm) and `googladtst.com` (which appears to be an impersonation of a Google service). On further investigation, using the RIPE Atlas stage, we found that Njalla's hosting is actually based near to Malmö and Copenhagen (the two cities are 8km apart).

Other interesting findings from this dataset include the fact that 21 (75%) of responsive IPs which report being in the Northern Mariana Islands are using incorrect geolocation, and that there is only one web server reporting to be in Christmas Island, and it uses correct geolocation.

5.2 Small Inter-Country IP Delegations

In this section, we will examine a much larger dataset. As mentioned in the Background chapter, the smallest routable prefix supported by external BGP is /24. This means that if two IP addresses are in the same /24, they will usually be in the same physical location. Therefore, if an IP delegation of a size smaller than /24 exists, and its parent netblock has a different country code, it is very likely that either the small netblock or its parent is using incorrect geolocation. We have analysed 100,000 randomly selected IP addresses out of a list of every web server in netblocks matching these criteria.

Figure 5.3 shows the results of this analysis. 92,351 IPs in this dataset were responsive, out of which 70,434 (76.3%) were using incorrect geolocation. 6.4% were using demonstrably correct geolocation, and 17.2% were inconclusive.

Overall, we identified 169 different country codes being used incorrectly, out of the 249 possible ISO-3166 countries. Figure 5.4 shows the 30 countries with most incorrect IP addresses. The country most impersonated is China, with 36,330 IP addresses advertising being in China, but actually being hosted elsewhere. Of these addresses, the majority (at least 32,680) are actually hosted in the US or Mexico. 78.8% of these IPs pretending to be in China are in netblocks advertised by AS54600 ("Peg Tech Inc.", a.k.a "raksmart.com"), an American-Chinese organisation offering web hosting.

Another interesting finding was that 18,850 incorrect IP addresses were in netblocks that contained "OVH" in their netblock name, suggesting the IPs are hosted by OVH, a French cloud hosting company. We discovered that OVH allows its customers to change the advertised geolocation of their web servers, in order to improve SEO results [54]. They offer their customers 13 different European country codes to use. For example, 57% of IPs with incorrect UK country code are hosted by OVH.

Figure 5.5 shows the top 20 hosting companies by number of incorrect IP addresses in this dataset. Some of the hosting companies represented in this list are large, well-known hosting companies which offer SEO services – for example, OVH, ettnet.se, and misshosting.com. However, many companies represented are not large or well-known, and may be companies that are hosting fraudulent content.

5.3 Tor Relays

In this section, we examine all IPs used by Tor relays, as of the 1st June 2020. There were 6083 such IPs.

We found that 6.7% of the relays in the Tor network are using incorrect IP geolocation. Among the 1329 exit relays, 9.3% of them were using incorrect geolocation. The top 10 countries that Tor relays pretended to be in were the United States (198), France (46), Panama (39), Singapore

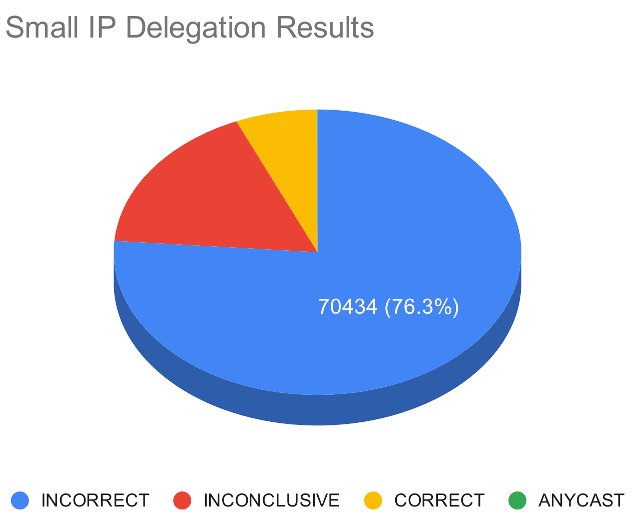


Figure 5.3: Breakdown of results from responsive IP addresses in the set of web servers in the Small Inter-Country IP Delegations dataset discussed in Section 5.2.

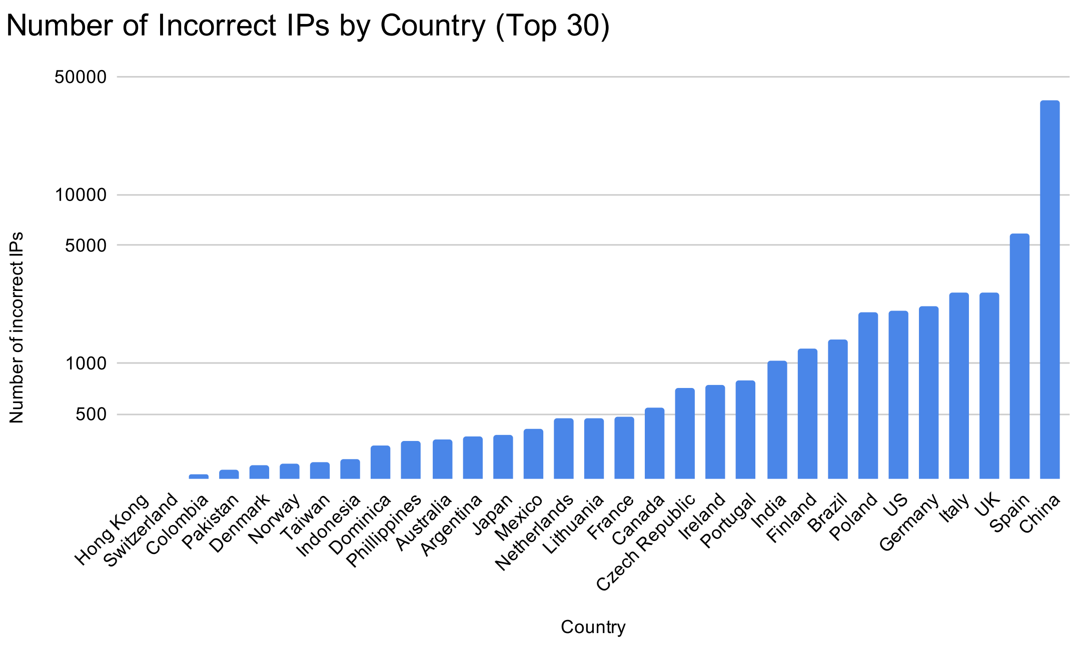


Figure 5.4: Number of incorrect IP addresses identified by advertised country (top 30 shown), in the Small Inter-Country IP Delegations dataset discussed in Section 5.2.

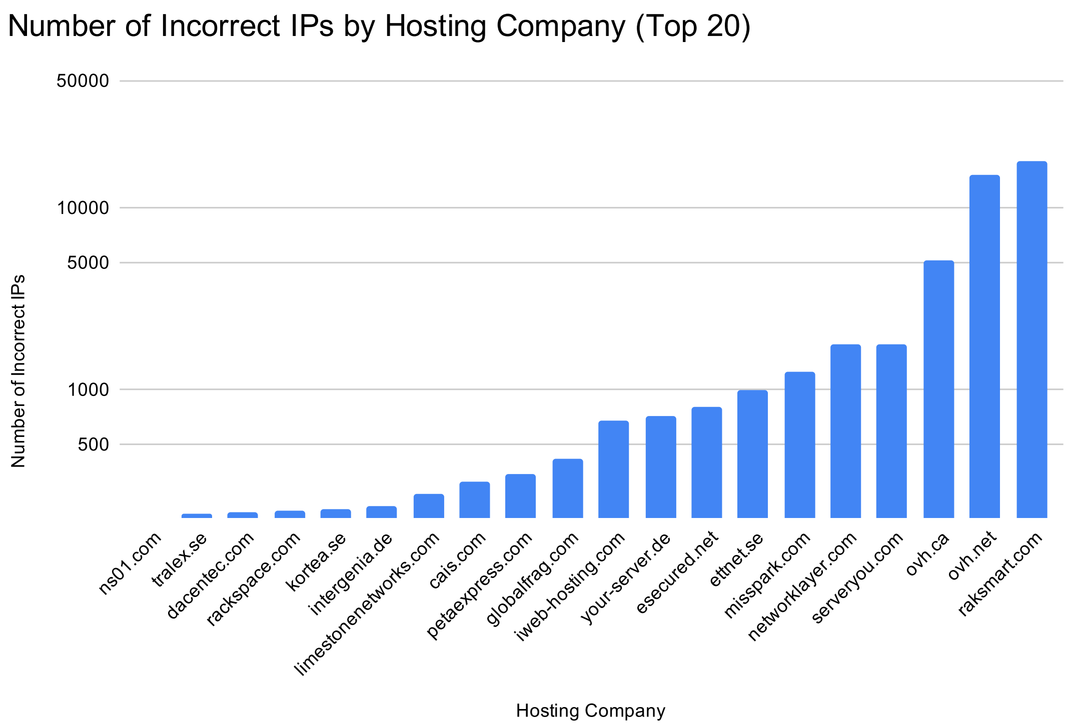


Figure 5.5: Number of incorrect IP addresses identified by hosting company (top 20 shown), in the Small Inter-Country IP Delegations dataset discussed in Section 5.2.

(17), Great Britain (13), Belize (12), Germany (9), Sweden (7), the Seychelles (6) and Russia (6). There are a relatively large number of relays pretending to be in countries with relaxed legislation such as Panama, Belize and the Seychelles. Four Tor relays are pretending to be in Bouvet Island. One is pretending to be in St Kitts and Nevis, a pair of tropical islands in the Caribbean.

5.4 VPN Providers

Finally, we have examined all IPs used by six VPN providers – HideMyAss (owned by Avast), PureVPN and VyprVPN, ExpressVPN, Surfshark, and Le-VPN. This dataset includes the "North Korean" VPN server mentioned in the introduction to this report. In total, this dataset consists of 2664 IP addresses, which is 17% more than the number considered in *How To Catch When Proxies Lie* [14].

We found that 738 out of the 2664 IPs (27.7%) used by these providers were using incorrect IP geolocation. The provider using most incorrect geolocation was VyprVPN, with 76.7% of its IPs being in netblocks with incorrect country codes. The provider using the most incorrect locations was HideMyAss, which uses incorrect IP geolocation targeting 180 different countries. Figure 5.6 shows a comparison of the prevalence of incorrect geolocation amongst each provider.

In total, 203 different country codes were used incorrectly in this dataset (there are 249 possible country codes). Figure 5.7 shows a map of all countries being used incorrectly by these three providers. Every South American country with the exception of French Guinea was being used incorrectly at least once. All but 4 African countries were being used incorrectly – the exceptions being Sierra Leone, South Africa, South Sudan and Western Sahara.

How To Catch When Proxies Lie found a very similar amount of incorrect geolocation to us – they were able to identify 28.1% of addresses as using incorrect geolocation. The fact that we are able to reproduce the results of their client-dependent system is encouraging evidence for the capability of our client-independent system.

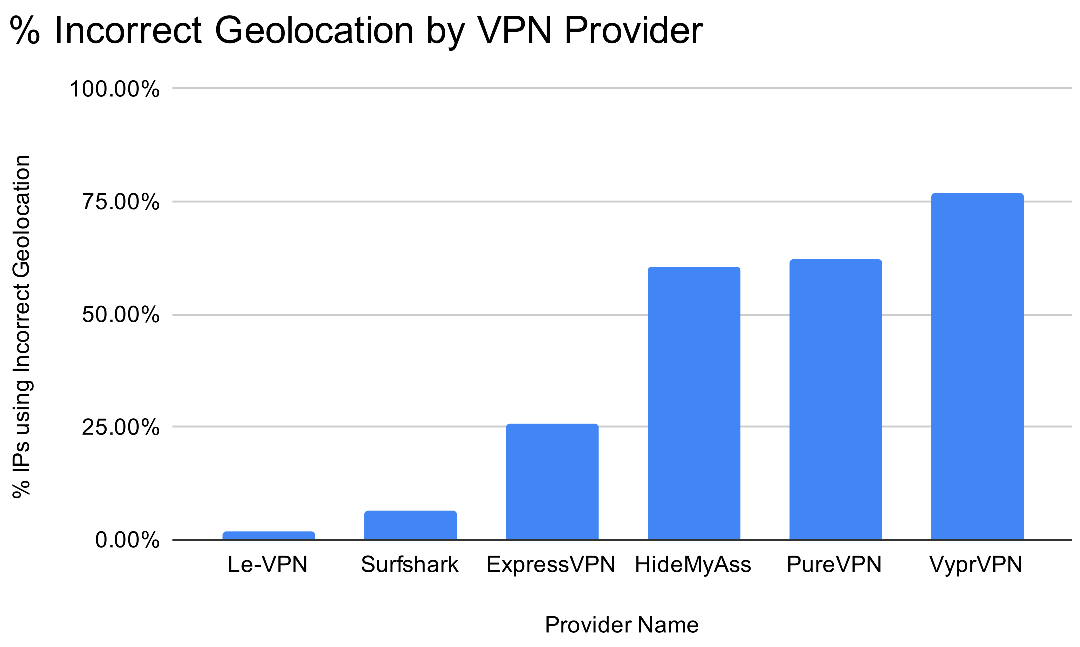


Figure 5.6: Percentage of IP addresses using demonstrably incorrect geolocation for each VPN provider discussed in Subsection 5.4.

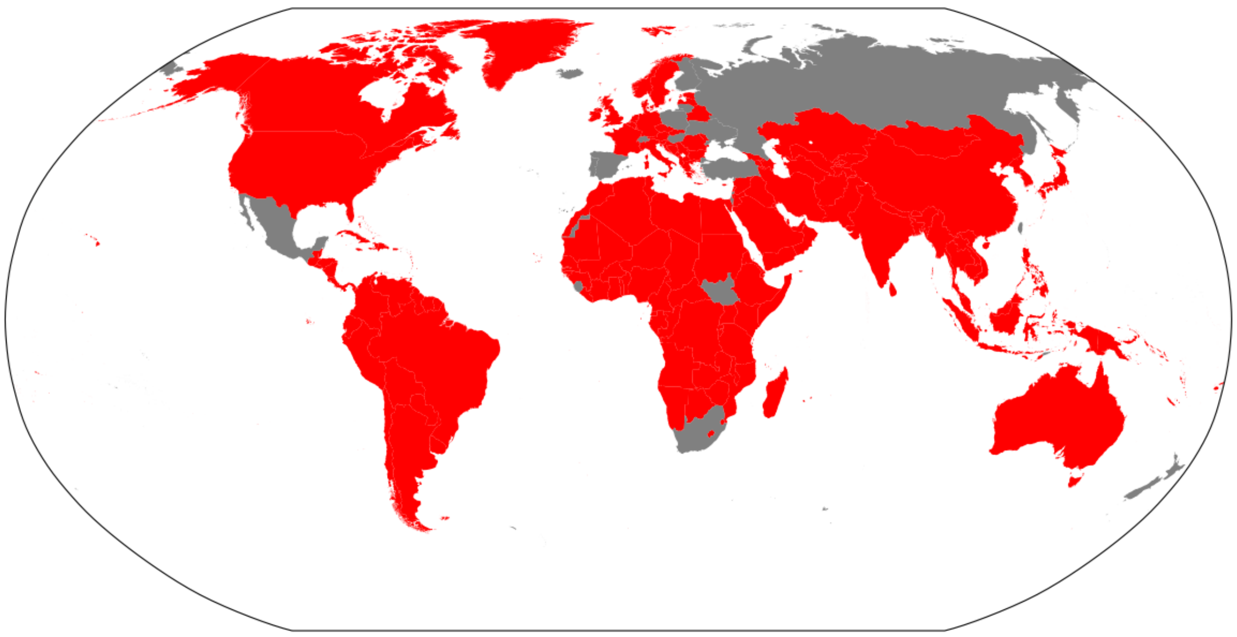


Figure 5.7: Map showing each country being used incorrectly by the six VPN providers described in Subsection 5.4. Countries filled in red are countries which have been used incorrectly in at least one case.

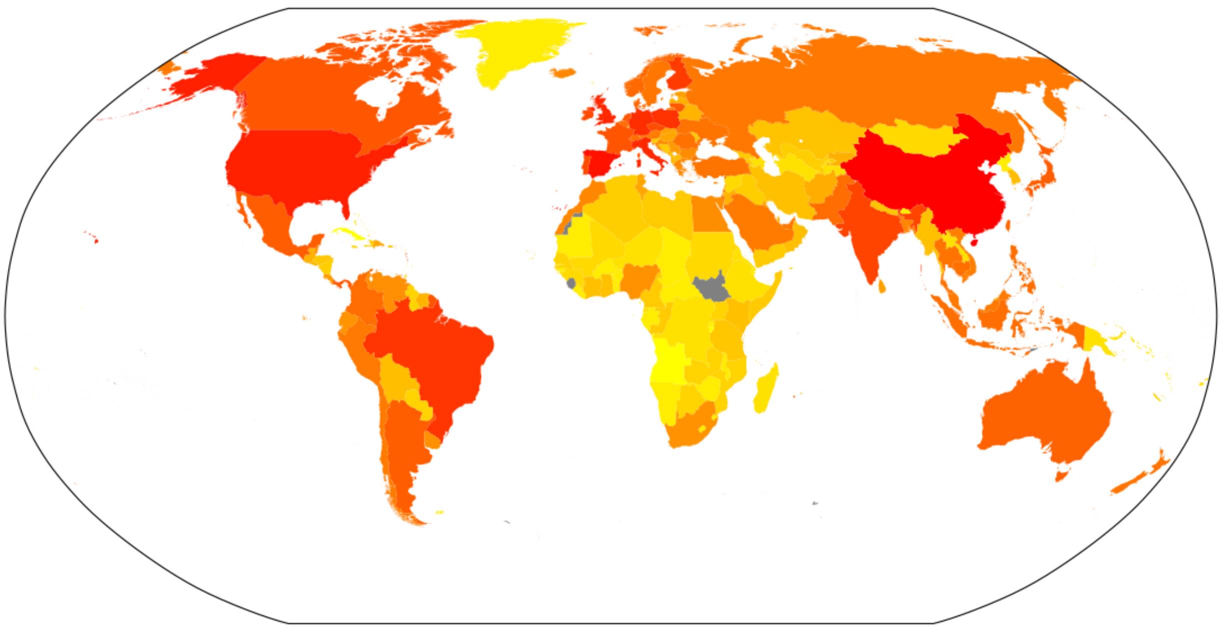


Figure 5.8: Heatmap showing the prevalence of IP geolocation fraud by targeted country, over all datasets examined in this report. Grey countries are countries where no geolocation fraud targeted that country. Colors have been calculated on a logarithmic scale, with the reddest countries having the most IP addresses targeting them, and the yellowest having the least.

Chapter 6

Evaluation

In this chapter, we will evaluate the work conducted in this project. We will start by evaluating the system as a whole, and will then evaluate our work on Caliper, the novel geolocation algorithm created in this project, separately.

6.1 Fraud Detection Pipeline Evaluation

The fraud detection pipeline designed – the core of our fraud detection system – works well. the results given in Chapter 5 are evidence of this. Our system is applicable to the real world, and has been proven to be able to analyse large batches of IPs. We've been able to gather data on the subject of IP geolocation fraud on a scale which, to the best of our knowledge, has not been described in the literature before. The only existing work we could find on IP geolocation fraud, *How To Catch When Proxies Lie* [14], studied only 2,269 individual IP addresses, whereas we have considered over 111,000.

Figure 6.1 shows the progress of all 111,108 unique IPs processed. The system was able to make a conclusive decision (either correct or incorrect) for 80,420 (72.4%) of the IPs submitted.

6.1.1 Throughput

Table 6.1 shows statistics about the time taken per IP for each of the datasets described in Chapter 5. In each dataset we have considered, excluding the Tor node dataset, each IP has taken on average between 1.15 and 6.75 seconds. This is a throughput of between 533 and 3130 IPs processed per hour.

The throughput of the system is dependent on the IP addresses being submitted. If a set of IP addresses is submitted with IPs in many unique /24, these IPs will take longer to analyse than a set of IP addresses where many of the IPs are in the same /24 blocks as each other, due to the caching logic. This is demonstrated by the throughput observed in the Tor node dataset, in which almost every IP was in a unique /24. In this dataset, the throughput was much lower – each IP took on

Batch Name	**IPs**	**Distinct /24s**	**Time Taken**	**Credits Used**	**Time/IP**	**Credits/IP**
Tax Haven IPs	6,247	373	2 hours	128,800	1.15s	20.6
Unlikely Countries	515	73	36 minutes	24,750	4.19s	48.1
Small IP Delegations	100,000	12,083	7 days	6,013,320	6.05s	60.1
Tor Nodes	6,083	4,770	3 days	1,956,280	42.0s	321.6
VPN Providers	2,664	859	5 hours	285,500	6.75s	107.2

Table 6.1: Statistics on throughput and credit expenditure for the batch datasets processed by the system. The *Distinct /24s* column shows the number of distinct /24 blocks present in the dataset.

average 42 seconds to process, giving a throughput of only 72 IPs per hour.

One bottleneck on the throughput of the system is the limitation of at most 100 parallel measurements imposed by the RIPE Atlas API. We have found that limiting access to at most 8 threads at once using a semaphore avoids hitting this limit, but this reduces the number of IPs which can be processed in parallel. Another bottleneck is the 1 million credits per day spending limit. This limit meant that during running the 100,000 IP dataset described in Chapter 5, we had to stop processing and wait for the credit limit to refresh before continuing.

6.1.2 Cost Analysis

The system has a number of static costs, but also has some load-dependent costs. The servers running `jping-api` each cost $5 per month, and we have used 8 of them, meaning there is an ongoing cost of $40 per month that the system is running. We obtained 50 million RIPE Atlas API credits free of charge for educational purposes, but normally they have to be obtained through sponsorship or probe hosting. A 1000€/year sponsorship to RIPE Atlas gives the sponsor the option of adopting 10 probes, which will earn 21,000 credits per day, or 7.7 million credits per year. This is an exchange rate of around 7700 credits/€. The amount of credits earned can be doubled if the sponsor hosts the probes for themselves, but the exchange rate remains quite low.

In terms of cost per IP address, in the set of 100K IP addresses considered in Section 5.2, 5472 IPs were submitted to the RIPE Atlas stage (5.5%). If the RIPE Atlas stage is needed for an IP, between 400 to 1600 credits are required, depending on the number of iterations. Thus, assuming other IPs processed in future will be similar to those encountered in this dataset, each IP will cost on average 22 to 88 credits, which translates to 0.28 to 1.12 cents.

Alternatively, organisations can earn RIPE Atlas credits by hosting a RIPE Atlas anchor and a set of probes. Our understanding [1] is that if the organisation was hosting five probes, and one anchor, they would earn $210,600 \times 6 = 1.26$ million credits per day, or roughly 113 million credits every 3 months. In the set of 100K addresses, each cached /24 block cost on average 498 credits. Assuming future IPs submitted cost similar to this, and assuming cache entries are refreshed every 3 months, the organisation could maintain a cache of 227K /24 blocks, spanning 58 million IP addresses.

6.1.3 RIPE Stage Evaluation

The RIPE stage of our system is performed in a number of iterations. Each iteration collects new measurements. The RIPE stage finishes when 3 iterations have been completed, or when the IP's geolocation veracity can be determined conclusively.

The goal of the RIPE stage is to verify the target's claimed geolocation, while spending as few RIPE Atlas credits as possible. To evaluate whether the RIPE stage is achieving this objective, we have recorded the progress of 519 IPs submitted to the stage, and we have drawn a Sankey diagram to chart their progress. From this chart, it appears that the stage that gets the most conclusive results is the first iteration. The later stages appear less likely to assist in a conclusive decision being made. In particular, very few IPs are found to be conclusive in the second iteration. Further work could be done on optimising the number of probes selected at each iteration, in order to minimise the number of probes used.

6.1.4 Detection Evasion

There are a number of ways that IP addresses can evade detection by IP geolocation systems. Our system has been built with resilience to these attacks in mind.

A number of other ways to avoid latency-based geolocation efforts exist. One such way is described in [55]. This approach relies on the timestamps which many ping tools attach to their ICMP packets. These tools send the ICMP packets with the current timestamp attached. The host

[1]Based on the credit system described at https://atlas.ripe.net/docs/credits/

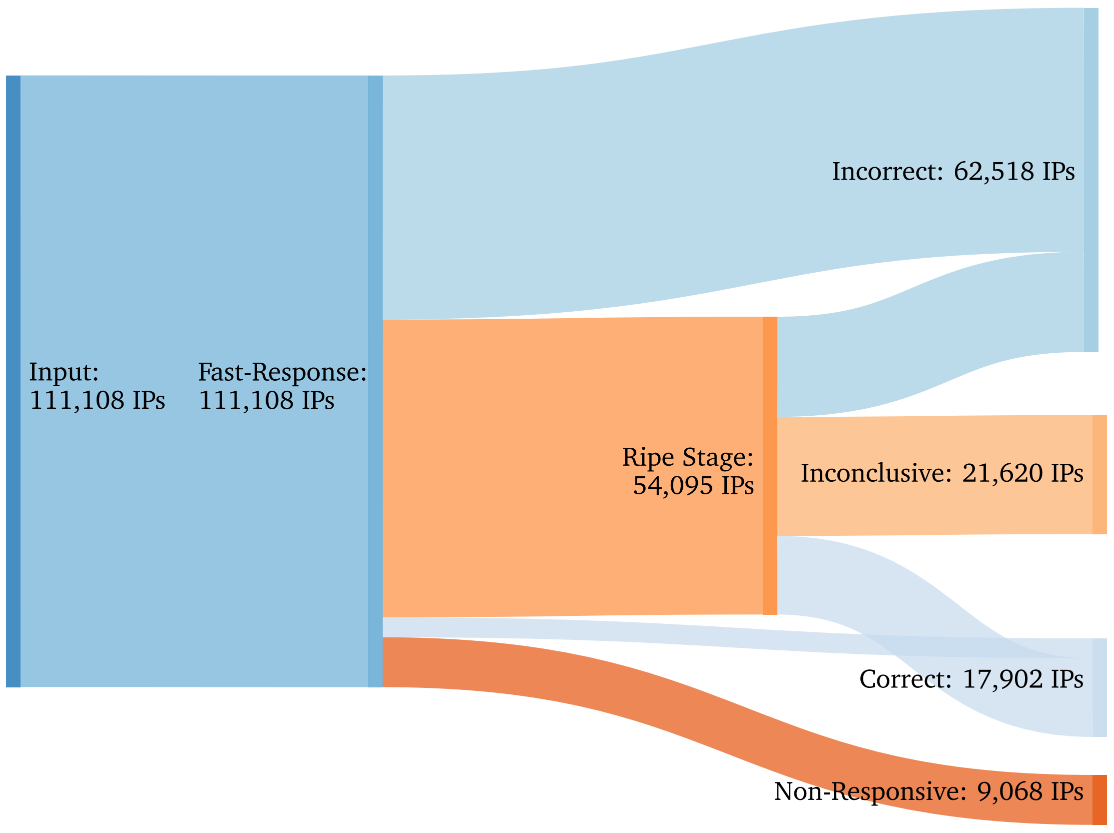


Figure 6.1: Sankey diagram showing the progress of all IP addresses processed through all stages.

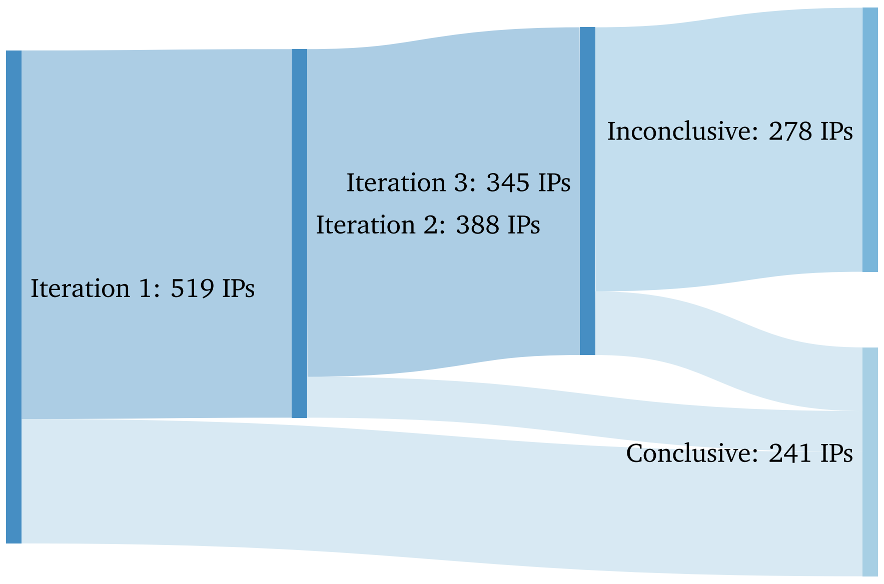


Figure 6.2: Sankey diagram showing the progress of a sample of 519 IP addresses going through the RIPE stage.

	197.231 .221.1	**5.62 .61.64**	**165.231 .147.103**	**199.191 .51.17**	**103.140 .242.8**	**172.111 .131.1**	**37.230 .177.2**
Actual Location	SE/DK	CZ	SE	US/MX	HK	BR	US
WHOIS Country	LR	KP	SC	VG	KY	PE	IT
MaxMind GeoIP2	LR	CZ	RU	VG	KY	PE	US
ipinfo.io	LR	CZ	EE	VG	US	PE	US
ipdata	LR	CZ	RU	VG	KY	PE	US
ipstack	LR	KP	SC	VG	HK	PE	IT
Hacker Target	LR	CZ	RU	VG	KY	PE	US
UltraTools	LR	KP	SC	VG	HK	PE	IT

Table 6.2: Results from existing commercial IP geolocation services for a range of IP addresses using fraudulent geolocation. Actual location is the location determined by our system – plots of the exact region each IP could be within can be found in Appendix Chapter A.

should then respond with the same timestamp, if the ICMP specification [23] is being followed. This timestamp is used to calculate network delay. However it's possible that the host can edit this timestamp before sending the ICMP response, meaning the latency measured by the ping tool becomes attacker-controlled. The attacker can then trick delay-based geolocation algorithms into giving outright incorrect results. Our system is not vulnerable to this attack, since `jping` does not rely on timestamps embedded in its ICMP packets. The ICMP packets sent by RIPE Atlas probes also do not contain timestamps in the packets (we initiated measurements to a server controlled by us, and examined the structure of the ICMP packets sent by the probe using `tcpdump -x`), so these measurements aren't vulnerable to this attack either.

Another way to evade detection would be to insert delays before replying to our ICMP Echo requests, as described in [52]. This kind of evasion cannot make a geolocation system give an *incorrect* solution, since the evading target can only pretend to be further away from monitors, and not pretend to be closer. However, it can make results less specific. Our system, like any geolocation system relying on delay measurements, can be made less specific using this approach.

A final way to evade detection is for attackers to configure firewalls, or hosts themselves, to drop ICMP packets. Our system is partially resilient to this attack. If the target doesn't respond to ICMP pinging, we instead use TCP measurements from the `jping-api` servers, which are more likely to elicit a response from the target, especially if it is a web server. However, the RIPE Atlas API doesn't support TCP pinging, so this stage cannot be applied to ICMP–unresponsive IPs.

6.1.5 Reliability of Results

Because the information generated by our system is designed to be used by internet security companies, or even law enforcement, it is critical that our results are reliable. By reliable, in this context we mean that if we determine that an IP has incorrect geolocation information, it must actually have incorrect geolocation information. Our system has been built with this in mind, and relies only on the Constraint-Based Geolocation algorithm to make decisions, an algorithm which is conservative in its calculations – the relationship between latency and distance it uses must not be violated by any of the training data. Because of this, the automatic decisions made by our system are reliable.

We have been able to obtain similar results when examining VPN providers to the current state-of-the-art study in this area, *How To Catch When Proxies Lie* (Subsection 5.4). This study used a client-dependent system, which relies on co-operation of the targets. The fact that we can reproduce their results using a client-independent system is very encouraging – it suggests that our system is as reliable and precise as theirs, while being applicable on a far larger scale.

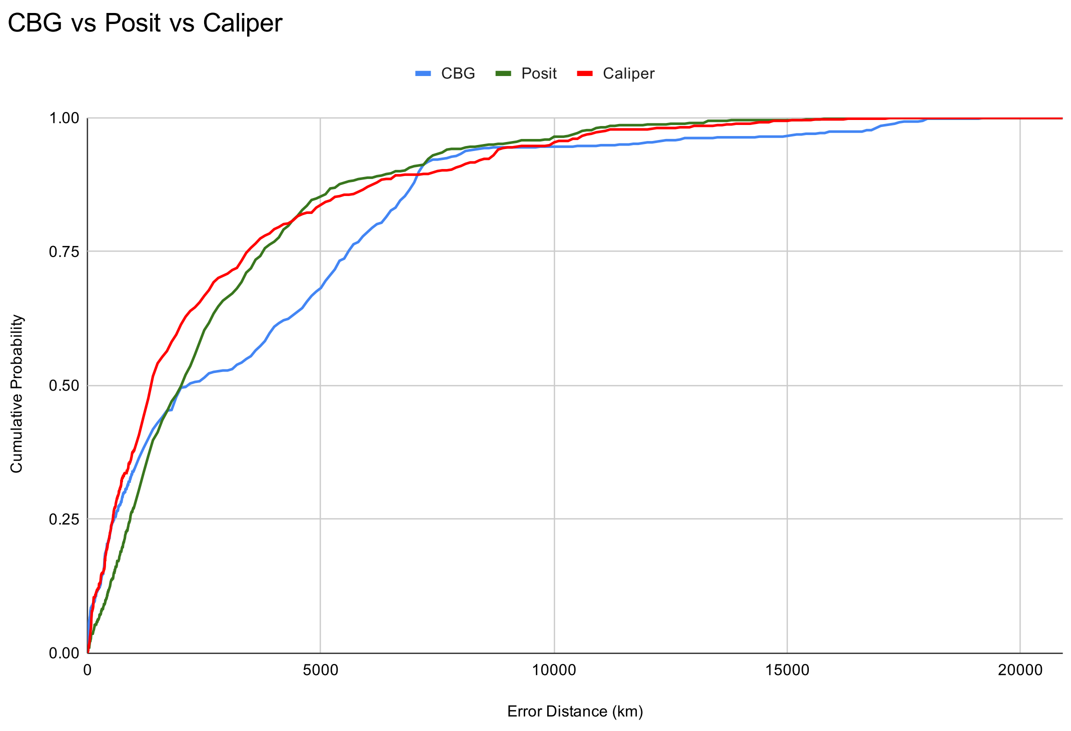


Figure 6.3: Comparison of CDF of geolocation error for testing dataset between CBG, Posit and Caliper. The cumulative probability for a given error distance is the proportion of IPs in the testing dataset that were classified with error less than that error distance.

6.1.6 Comparison to Commercial Geolocation Services

Table 6.2 shows the performance of existing, commercial IP geolocation tools on a number of IP addresses which are using fraudulent geolocation, compared to the actual location determined by our system. The results from this experiment show that while these tools might give reasonable answers in normal circumstances, they do not perform well when targets are deliberately using fraudulent geolocation. Our system is able to give better information on these kinds of targets, because none of our automatic decision making process is based on information controlled by the target.

6.2 Caliper Evaluation

In order to evaluate the performance of Caliper, we have written implementations of two other geolocation algorithms – CBG and Posit – to compare performance against. We will evaluate using the measures most commonly used in the literature when discussing geolocation algorithms – median error distance, and graphs of the cumulative distirbution of error distances in the testing dataset. Our implementation of CBG is exactly as described in [36]. Our implementation of Posit is as described in [40], with one exception – we have adapted Posit to use a global solution search, instead of only using monitor and landmark positions as candidate location. This is because we don't use a large number of landmarks and monitors, so Posit's performance was very poor when its original solution finding strategy (which only considers the position of landmarks and monitors) was employed.

The testing data described in Section 3.1 (747 data points in total) were used to perform the testing. Each target in the testing dataset was geolocated (using measurements collected and saved to disk), and the error for each data point was recorded. The tests were run on an 8-core machine, and a pool of 7 `multiprocessing` processes was used by both Caliper and Posit. 68 landmarks and 8 monitors were used in this analysis, which is quite a low number – as such, the absolute errors will be high, but the relative error between the algorithms is the most important metric to consider.

The median error for this dataset was 2121.4km for CBG, 2006.6km for Posit, and 1337.2km for Caliper. This confirms that Posit is able to outperform CBG, but also shows that our Caliper implementation performs significantly better than both. The code used to perform this testing can be found in `ip_geolocation.evaluation.evaluate_algorithms`.

A graph of the Cumulative Distribution Function of geolocation error recorded when geolocating this dataset can be found in Figure 6.3. From the graph, it can be seen that Caliper's location guesses are within 4000km significantly more often than both CBG and Posit. An inflection point can be seen at around 4000km, indicating that Posit has slightly less errors of greater than 4000km than Posit does. Both Posit and Caliper have significantly less errors greater than 10000km than CBG does. In 451 out of the 747 testing data points (60%), Caliper gives a lower geolocation error than Posit. This data justifies our decision to create and use a new geolocation algorithm, capable of better performance on a global scale.

6.3 Ethical Evaluation

We believe that we have been aware of the ethical implications of this project (as outlined in Section 1.3), and have taken steps to ensure that the research performed in this project has been done ethically. We have not performed geolocation on any end users – only VPN servers, web servers and Tor relays. We have actively chosen against examining peer-to-peer networks such as I2P, because it would be impossible to avoid geolocating end users in these cases.

We also believe that the results described in this report have a number of positive ethical implications – making VPN users better informed about the claims of VPN providers they are purchasing products from, making Tor users more aware of where their traffic may be exiting from, and demonstrating that internet security companies and law enforcement agencies should not blindly trust WHOIS data.

There are plans to use the system developed in this project to help identify servers and hosting companies serving fraudulent/illegal content – it is believed there may be a correlation between the usage of incorrect geolocation and fraudulent activity. When the system is used in this way, it will be contributing to the overall safety of internet users. We see this as a positive ethical contribution of this project.

Chapter 7

Conclusion

To conclude this report, we will summarise the achievements of this project, and give some ideas of where future work could be performed.

7.1 Summary

In this project, we have examined the problem of IP geolocation fraud, and have designed a system capable of detecting this kind of fraud on a large scale. In the process of developing this fraud detection system, we have also described a number of other new geolocation tools – Caliper, a new statistical geolocation algorithm which outperforms other existing approaches, `jping`, a new ping tool written with parallelism and tamper-resistance in mind, and a web interface for presenting results from the system in a way that could be used by an internet security company or law enforcement agency.

As mentioned in the abstract, and demonstrated in Chapter 5, we have analysed the prevalence of geolocation fraud in address space spanning over 4 million IPs, and have discovered over 62,000 individual IPs using incorrect geolocation. We've found 8,507 /24 blocks using incorrect geolocation, which equates to address space spanning greater than 2.1 million IPs.

The system and web interface presented in this project has been deployed and is in use at Netcraft, and will continue to be used after this project's conclusion. Plans are being developed to integrate the results of this system with Netcraft's takedown service – when implemented, users will be able to see whether servers hosting fraudulent content are using fraudulent IP geolocation.

We believe that each objective laid out in Section 1.1 has been achieved. Objective 1a) is achieved by the fraud detection pipeline; Objective 1b) by the web interface; and Objective 1c) by the batch processing feature. Chapter 5 achieves Objective 2.

7.2 Future Work

7.2.1 Improve Caliper Solution Search Method

Currently, Caliper uses a rudimentary grid search in order to find an optimal candidate solution. Future work could be conducted to research more sophisticated, and more efficient, methods of optimisation. These methods could then be used on statistical geolocation algorithms developed in the future.

7.2.2 Use More Servers in Fast-Response Stage

Future researchers looking to implement similar systems might consider using a larger set of Fast-Response monitors in order to limit the number of RIPE Atlas credits needed. It isn't clear exactly what the relationship between the number of Fast-Response monitors and the number of IPs entering the RIPE stage is, but we imagine that the more Fast-Response monitors, the less IPs will need the RIPE stage.

7.2.3 Determine Optimal Caching Time

Analysis of how often IP geolocation results should be cached for – essentially, finding how often servers in a netblock change their location on average – would be a useful piece of future research. Another idea would be to periodically check whether RIPE stage cache entries are sane using the cheaper Fast-Response stage of the system. The RIPE stage entry should be within the region found by the Fast-Response stage – if it isn't, the servers will be in a different physical location since the cache entry was created, and the cache entry should be refreshed.

7.2.4 Trust Reputable Hosters

Future researchers could consider filtering out IP addresses in netblocks owned by legitimate entities (such as Google, DigitalOcean and Amazon), who will not be using fraudulent geolocation, in order to reduce the number of lookups their system needs to do.

7.2.5 Use BGP & Routing Information

BGP information – information about how packets are routed between Autonomous Systems (ASs) on the internet – could be used as additional information to use while performing geolocation fraud analysis. For example, if an IP address is in a netblock announced by an AS in Sweden, which peers only with other ASs in Sweden, it is very likely that the host with that IP is itself in Sweden.

Another idea for using routing information could be building up a graph of intermediate routers and their countries. This could be achieved by performing traceroutes to IPs with known location, and assigning countries to intermediate routers based on the location of the end hosts that they route packets towards – for example, an intermediate router that is only seen forwarding packets to hosts in Germany is probably in Germany. This could then be performed recursively, in order to build up a graph of intermediate routers and their countries. We've not had time to investigate this idea in this project, but future work on this would be interesting.

7.2.6 Use Probabilistic Answers

In cases where the Fast-Response and RIPE stages were inconclusive, Caliper, or a different geolocation algorithm, could be used to produce a list of feasible countries with probabilities that the target is in each of those countries.

A simple way to do this would be to consider the areas of each country that are included in the feasible region. For example, if the feasible region is mostly in the UK, but just about clips the north part of France, the target is more than likely in the UK. A more robust approach could be to adapt Caliper to be able to calculate the probabilities of given areas, rather than just given candidate locations. This could be done, for example, by integrating the likelihood function over each location in the given area.

Automatic decision making could be performed if, for example, the probability of it being in one particular country would be far higher than the probability of it being in any other country. Setting a safe threshold for this decision making would be important, in order to not negatively impact the reliability of results from the system.

Bibliography

[1] Doyle, C. C., Mieder, W. & Shapiro, F. R. (2012) *The Dictionary of Modern Proverbs*. Yale University Press, New Haven, Connecticut.

[2] Li, K. & Du, T. C., Building a targeted mobile advertising system for location-based services. *Decision Support Systems*, 54 (1), (2012), 1–8. doi:10.1016/j.dss.2012.02.002.

[3] Morgan, J., Ong, D. & Zhong, Z. Z., Location still matters: Evidence from an online shopping field experiment. *Journal of Economic Behavior & Organization*, 146, (2018), 43–54. doi:10.1016/j.jebo.2017.11.021.

[4] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R. & Weihl, B., Globally distributed content delivery. *IEEE Internet Computing*, 6 (5), (2002), 50–58. doi:10.1109/mic.2002.1036038.

[5] Quah, J. T. S. & Sriganesh, M. (2007) Real Time Credit Card Fraud Detection using Computational Intelligence. In: *2007 International Joint Conference on Neural Networks*, pp. 863–868.

[6] Gowtham, R. & Krishnamurthi, I., A comprehensive and efficacious architecture for detecting phishing webpages. *Computers & Security*, 40, (2014), 23–37. doi:10.1016/j.cose.2013.10.004.

[7] Cross, C., 'Oh we can't actually do anything about that': The problematic nature of jurisdiction for online fraud victims. *Criminology & Criminal Justice*. doi:10.1177/1748895819835910.

[8] Access to Databases Critical to Enforce Consumer Protection Laws. Available from: https://www.ftc.gov/news-events/press-releases/2006/06/ftc-issues-statement-whois-databases. [Accessed: 21st January 2020].

[9] HideMyAss. See All Our Server Locations | HideMyAss! Available from: https://www.hidemyass.com/servers. [Accessed 18th January 2020].

[10] Daigle, L. (2004). WHOIS Protocol Specification. RFC 3912. doi:10.17487/RFC3912.

[11] MaxMind LLC. GeoIP Databases & Services: Industry Leading IP Intelligence | MaxMind. Available from: https://www.maxmind.com/en/geoip2-services-and-databases. [Accessed 18th January 2020].

[12] ipdata.co. IP Geolocation API with Threat Intelligence. Available from: https://ipdata.co/. [Accessed 18th January 2020].

[13] ipinfo.io. IPInfo.io: The Trusted Source for IP Address Data. Available from: https://ipinfo.io/. [Accessed 18th January 2020].

[14] Weinberg, Z., Cho, S., Christin, N., Sekar, V. & Gill, P. (2018) How to Catch When Proxies Lie: Verifying the Physical Locations of Network Proxies with Active Geolocation. In: *Proceedings of the Internet Measurement Conference 2018*, p. 203–217. doi:10.1145/3278532.3278551.

[15] Collins English Dictionary: Geolocation. Available from: https://www.collinsdictionary.com/dictionary/english/geolocation. [Accessed 21st January 2020].

[16] Housley, R., Curran, J., Huston, G. & Conrad, D. R. (2013). The Internet Numbers Registry System. RFC 7020. doi:10.17487/RFC7020. Available from: https://rfc-editor.org/rfc/rfc7020.txt.

[17] Fuller, V. & Li, T. (2006). Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632. doi:10.17487/RFC4632. Available from: https://rfc-editor.org/rfc/rfc4632.txt.

[18] CloudFlare. The Weird and Wonderful World of DNS LOC Records. Available from: https://blog.cloudflare.com/the-weird-and-wonderful-world-of-dns-loc-records/. [Accessed 21st January 2020].

[19] Hawkinson, J. A. & Bates, T. J. (1996). Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930. doi:10.17487/RFC1930. Available from: https://rfc-editor.org/rfc/rfc1930.txt.

[20] Dingledine, R., Mathewson, N. & Syverson, P. (2004) Tor: The Second-Generation Onion Router. In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, p. 21. USENIX Association, USA.

[21] Laki, S., Matray, P., Haga, P., Csabai, I. & Vattay, G. (2009) A detailed path-latency model for router geolocation. In: *2009 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops*, pp. 1–6. doi:10.1109/TRIDENTCOM.2009.4976258.

[22] Ramaswamy, R., Ning Weng & Wolf, T. (2004) Characterizing network processing delay. In: *IEEE Global Telecommunications Conference, 2004.*, volume 3, pp. 1629–1634 Vol.3. doi:10.1109/GLOCOM.2004.1378257.

[23] Postel, J. (1981). Internet Control Message Protocol. RFC 792. doi:10.17487/RFC0792.

[24] Gupta, M. & Conta, A. (2006). Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443. doi:10.17487/RFC4443.

[25] RIPE NCC, Ripe Atlas: A Global Internet Measurement Network. *Internet Protocol Journal*, 18 (3).

[26] Moore, D., Periakaruppan, R., Donohoe, J. & Claffy, K. (2000) Where in the world is netgeo.caida.org? In: *International Networking Conference (INET)*.

[27] Spring, N., Mahajan, R., Wetherall, D. & Anderson, T., Measuring ISP Topologies with Rocketfuel. *IEEE/ACM Trans. Netw.*, 12 (1), (2004), 2–16. doi:10.1109/TNET.2003.822655.

[28] Huffaker, B., Fomenkov, M. & Claffy, K., DRoP. *ACM SIGCOMM Computer Communication Review*, 44 (3), (2014), 5–13. doi:10.1145/2656877.2656879.

[29] Dan, O., Parikh, V. & Davison, B. D., IP Geolocation through Reverse DNS. [Preprint] Available from: https://arxiv.org/pdf/1811.04288.pdf [Accessed: 14th January 2020].

[30] Guo, C., Wang, H., Zhang, Y. & Liu, Y. (2007) Mining the Web for IP Address Geolocations. Technical Report MSR-TR-2007-139.

[31] Netcraft (2013). Amazon Web Services' growth unrelenting. Available from: https://news.netcraft.com/archives/2013/05/20/amazon-web-services-growth-unrelenting.html. [Accessed 2nd June 2020].

[32] Katz-Bassett, E., John, J. P., Krishnamurthy, A., Wetherall, D., Anderson, T. & Chawathe, Y. (2006) Towards IP Geolocation Using Delay and Topology Measurements. In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, p. 71–84. Association for Computing Machinery, New York, NY, USA. doi:10.1145/1177080.1177090.

[33] Padmanabhan, V. N. & Subramanian, L., An investigation of geographic mapping techniques for internet hosts. *ACM SIGCOMM Computer Communication Review*, 31 (4), (2001), 173–185. doi:10.1145/964723.383073.

[34] Choi, B.-Y., Moon, S., Zhang, Z.-L., Papagiannaki, K. & Diot, C. (2004) Analysis of Point-To-Point Packet Delay In an Operational Network. In: *IEEE INFOCOM 2004*, pp. 1797–1807 vol.3. doi:10.1109/INFCOM.2004.1354590.

[35] Percacci, R. & Vespignani, A., Scale-free behavior of the Internet global performance. *The European Physical Journal B - Condensed Matter*, 32 (4), (2003), 411–414. doi:10.1140/epjb/e2003-00123-6.

[36] Gueye, B., Ziviani, A., Crovella, M. & Fdida, S., Constraint-based geolocation of internet hosts. *IEEE/ACM Trans. Netw.*, 16, (2006), 1219–1232. doi:10.1145/1028788.1028828.

[37] Sturm, J. F., Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software*, 11 (1-4), (1999), 625–653.

[38] Govindan, R. & Tangmunarunkit, H. (2000) Heuristics for Internet map discovery. In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 3, pp. 1371–1380 vol.3. doi:10.1109/INFCOM.2000.832534.

[39] Wong, B. & Stoyanov, I. (2007) Octant: A Comprehensive Framework for the Geolocalization of Internet Hosts. In: *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*. USENIX Association, Cambridge, MA. doi:10.5555/1973430.1973453.

[40] Eriksson, B., Barford, P., Maggs, B. & Nowak, R., Posit: A Lightweight Approach for IP Geolocation. *SIGMETRICS Perform. Eval. Rev.*, 40 (2), (2012), 2–11. doi:10.1145/2381056.2381058.

[41] Wang, Y., Burgener, D., Flores, M., Kuzmanovic, A. & Huang, C. (2011) Towards Street-Level Client-Independent IP Geolocation. In: *NSDI'11 Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pp. 365–379. USENIX Association Berkeley, CA, USA.

[42] Google (2020). HTTPS encryption on the web - Google Transparency Report. Available from: https://transparencyreport.google.com/https/overview?hl=en.

[43] Wang, Z., Li, H., Li, Q., Li, W., Zhu, H. & Sun, L., Towards IP geolocation with intermediate routers based on topology discovery. *Cybersecurity*, 2, (2019), 1–14.

[44] Felt, A. P., Barnes, R., King, A., Palmer, C., Bentzel, C. & Tabriz, P. (2017) Measuring HTTPS Adoption on the Web. In: *26th USENIX Security Symposium (USENIX Security 17)*, pp. 1323–1338. USENIX Association, Vancouver, BC.

[45] Youn, I., Mark, B. & Richards, D. (2009) Statistical Geolocation of Internet Hosts. pp. 1 – 6. doi:10.1109/ICCCN.2009.5235373.

[46] Eriksson, B., Barford, P., Sommers, J. & Nowak, R. (2010) A Learning-Based Approach for IP Geolocation. pp. 171–180. doi:10.1007/978-3-642-12334-4_18.

[47] Nazario, J. & Holz, T. (2008) As the net churns: Fast-flux botnet observations. In: *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 24–31.

[48] Castelluccia, C., Kaafar, M. A., Manils, P. & Perito, D. (2009) Geolocalization of proxied services and its application to fast-flux hidden servers. pp. 184–189. doi:10.1145/1644893.1644915.

[49] Banks, G., Fattori, A., Kemmerer, R., Kruegel, C. & Vigna, G. (2011) MISHIMA: Multilateration of Internet Hosts Hidden Using Malicious Fast-Flux Agents (Short Paper). pp. 184–193. doi:10.1007/978-3-642-22424-9_11.

[50] Abdou, A. & Oorschot, P. C. V., Server Location Verification (SLV) and Server Location Pinning: Augmenting TLS Authentication. *ACM Trans. Priv. Secur.*, 21 (1). doi:10.1145/3139294.

[51] Candela, M., Gregori, E., Luconi, V. & Vecchio, A., Using RIPE Atlas for Geolocating IP Infrastructure. *IEEE Access*, 7, (2019), 48816–48829.

[52] Gill, P., Ganjali, Y., Wong, B. & Lie, D. (2010) Dude, Where's That IP? Circumventing Measurement-Based IP Geolocation. In: *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, p. 16. USENIX Association, USA.

[53] International Organization for Standardization (2020). Online Browsing Platform (OBP). Available from: https://www.iso.org/obp/ui. [Accessed 2nd June 2020].

[54] OVH (2020). IPs and geolocation - OVH. Available from: https://www.ovh.co.uk/web-hosting/ip.xml. [Accessed 2nd June 2020].

[55] Abdou, A., Matrawy, A. & van Oorschot, P. C. (2017) Accurate Manipulation of Delay-Based Internet Geolocation. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, p. 887–898. Association for Computing Machinery, New York, USA. doi:10.1145/3052973.3052993.

A Feasible Region Plots

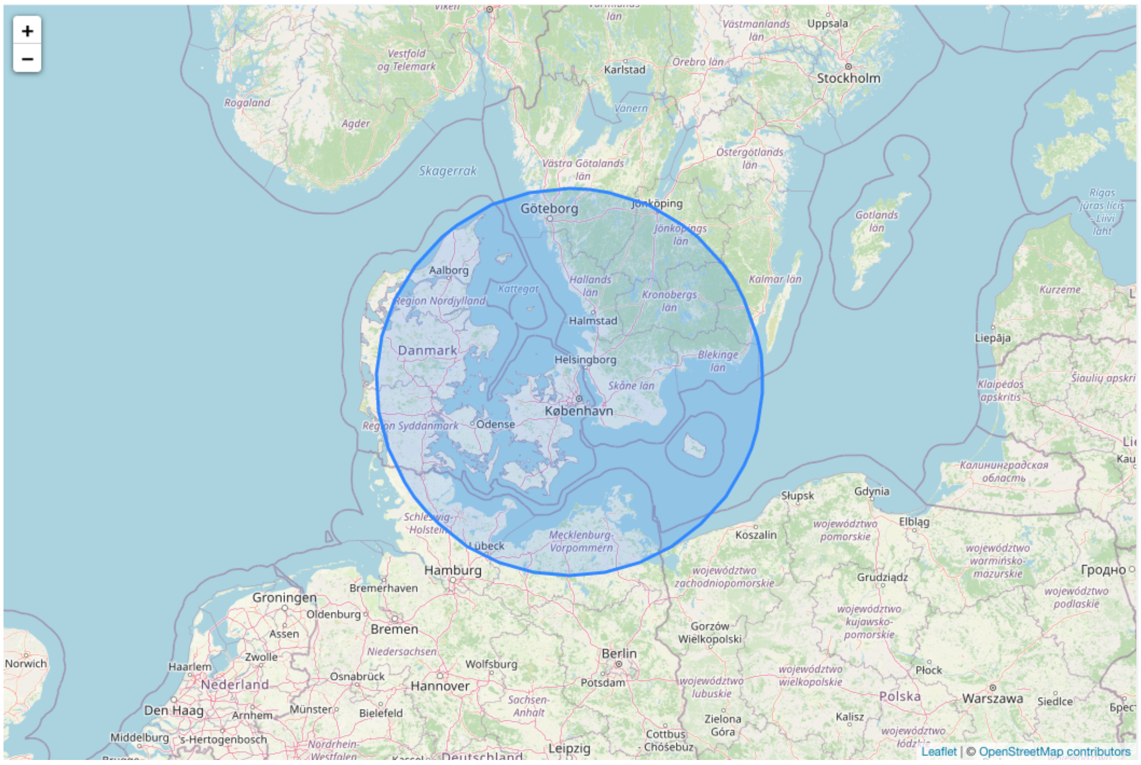


Figure A.1: Result of iterative CBG from our fraud detection pipeline on 196.231.221.1, an IP in the same “Liberian” netblock as the `Hydra1` Tor exit node.

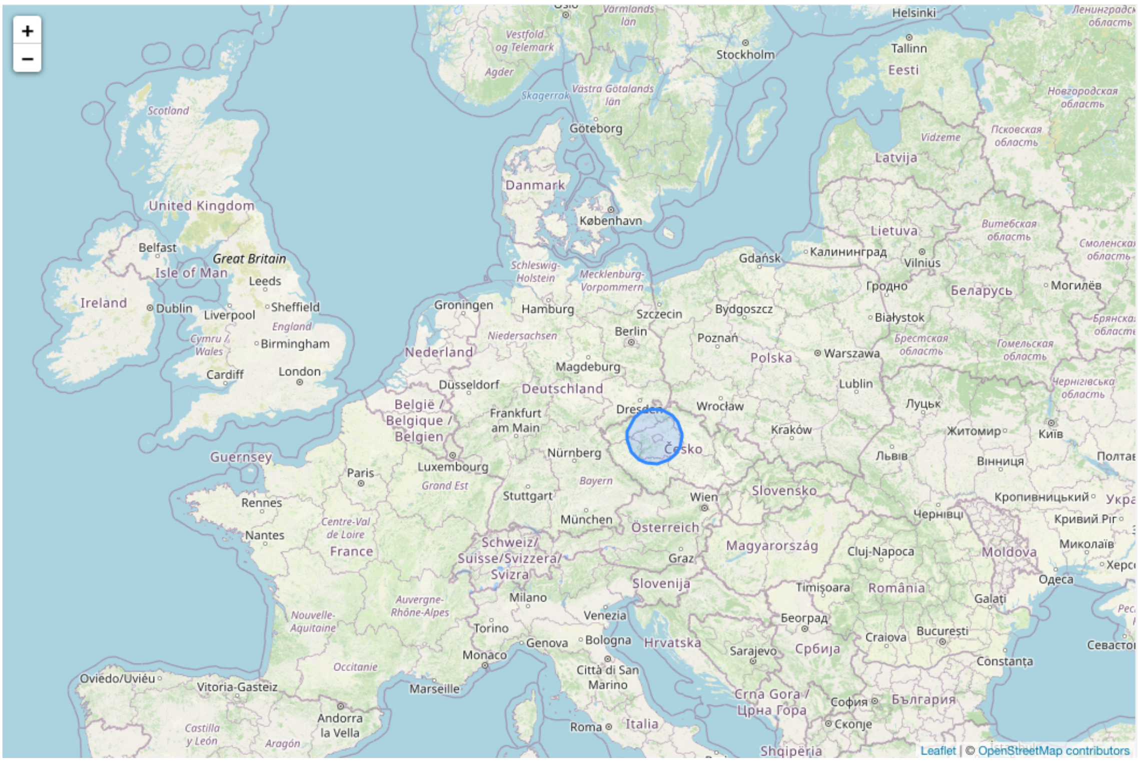


Figure A.2: Result of iterative CBG from our fraud detection pipeline on 5.62.61.64, which is in a netblock claiming to be in North Korea.

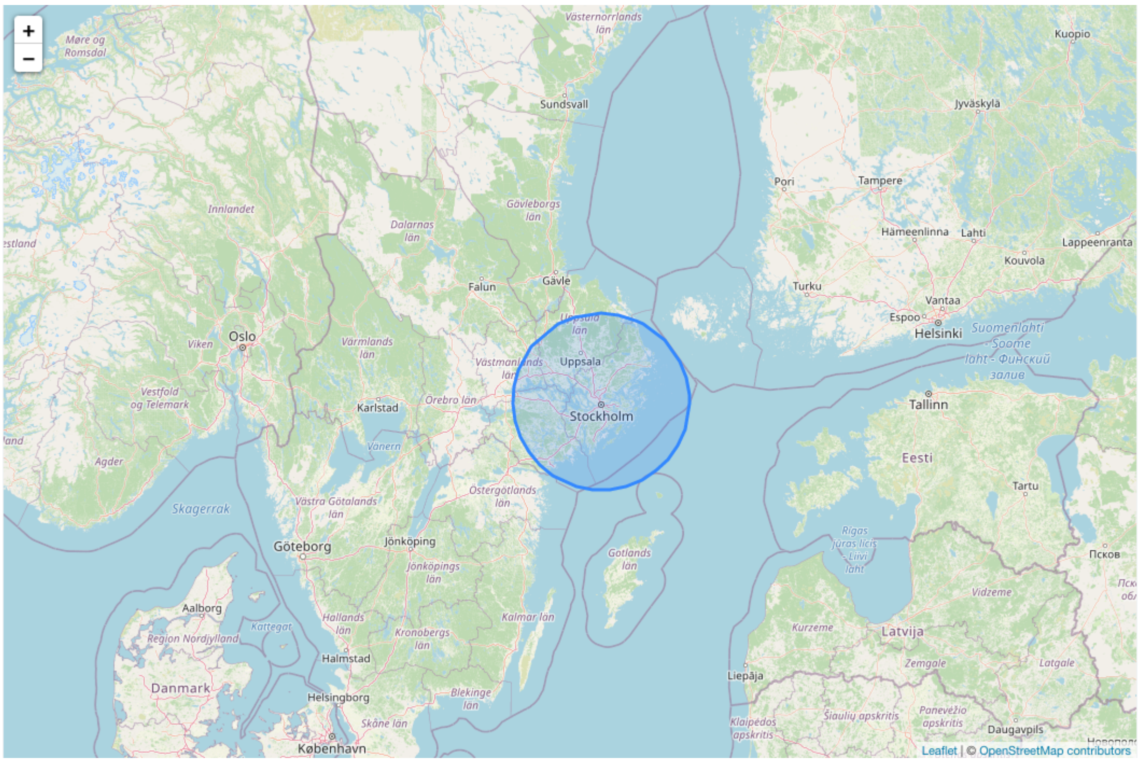


Figure A.3: Result of iterative CBG from our fraud detection pipeline on 165.231.147.103, which is in a netblock claiming to be in the Seychelles.

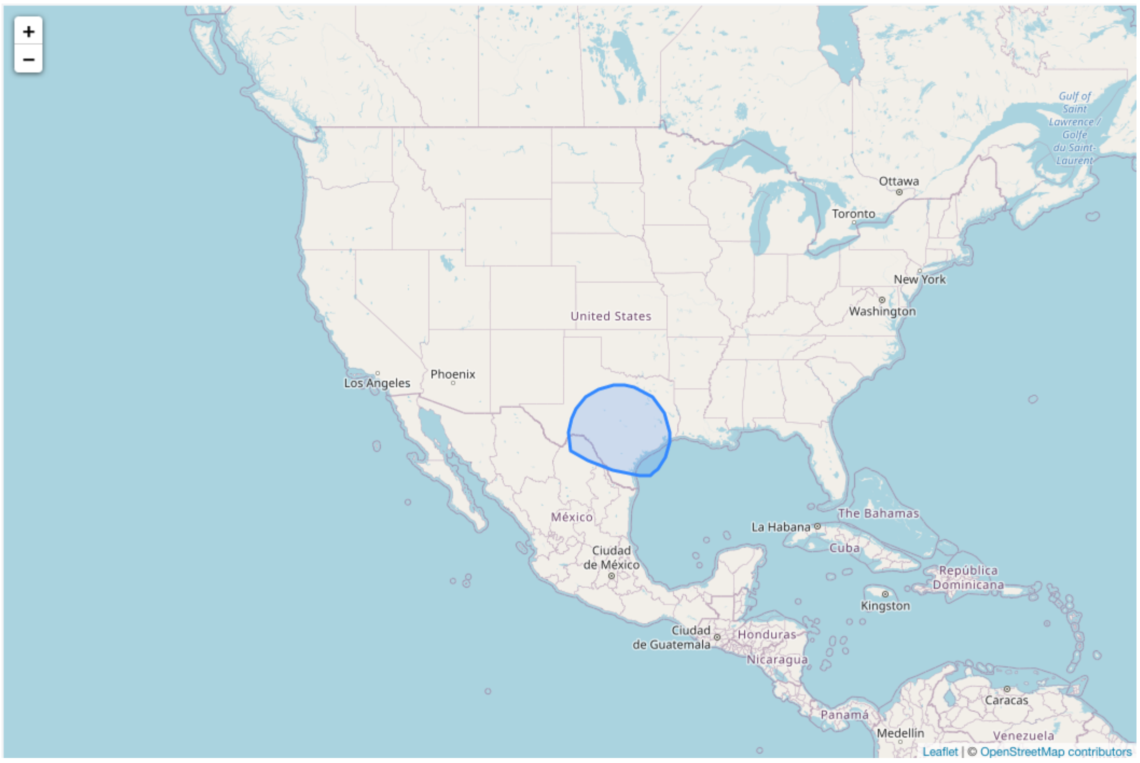


Figure A.4: Result of iterative CBG from our fraud detection pipeline on 199.191.51.17, which is in a netblock claiming to be in the Virgin Islands.

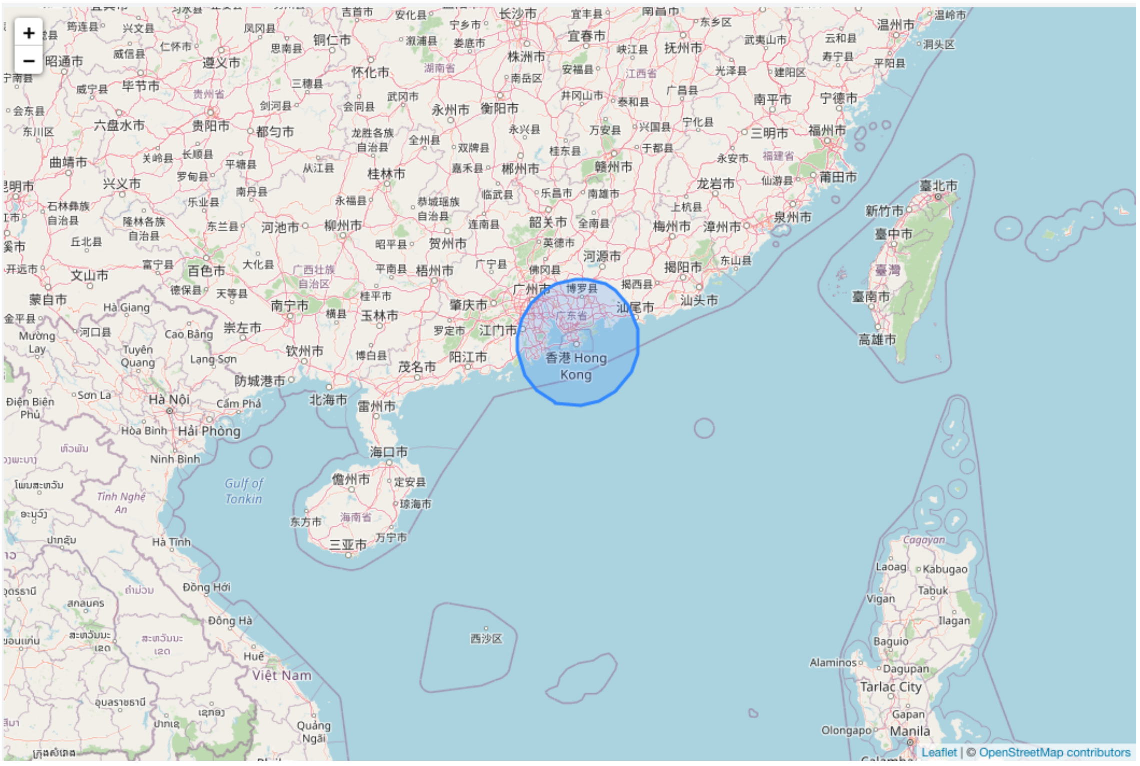


Figure A.5: Result of iterative CBG from our fraud detection pipeline on 103.140.242.8, which is in a netblock claiming to be in the Cayman Islands.

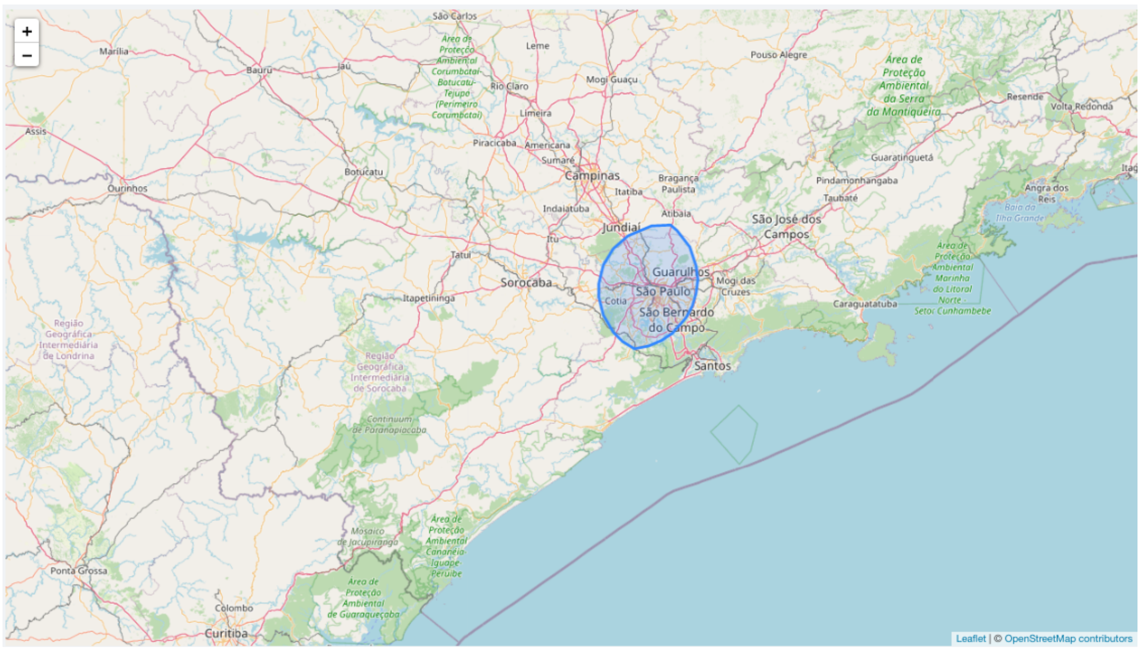


Figure A.6: Result of iterative CBG from our fraud detection pipeline on 172.111.131.1, which is in a netblock claiming to be in Peru (it in fact in Sao Paolo, Brazil).

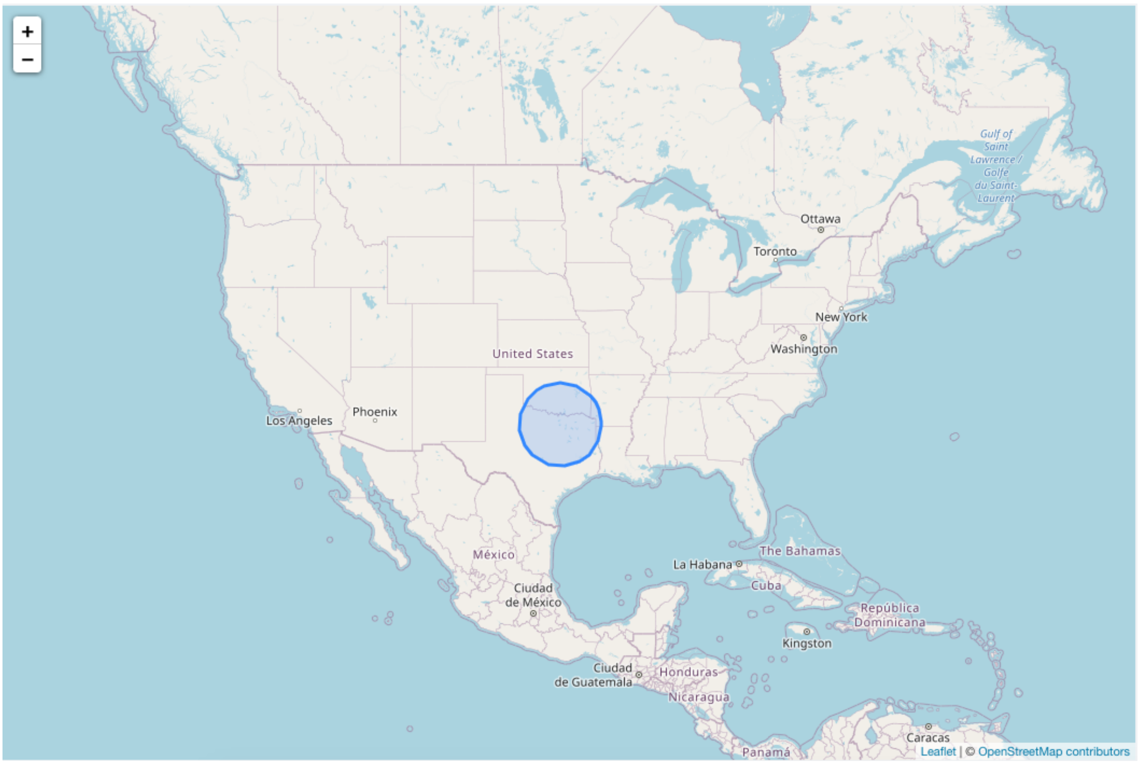


Figure A.7: Result of iterative CBG from our fraud detection pipeline on 37.230.177.2, which is in a netblock claiming to be in Italy.