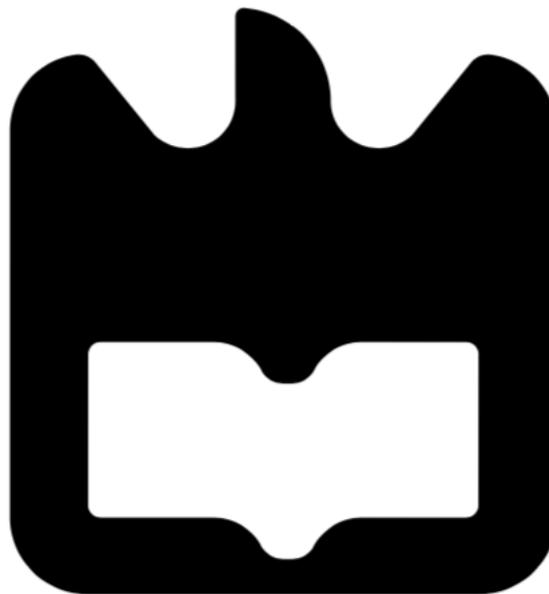




**José Miguel Guerra da
Fonseca**

Location of Internet Entities

Localização de Entidades da Internet





**José Miguel Guerra da
Fonseca**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Paulo Salvador, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professora Doutora Susana Isabel Barreto de Miranda Sargento

Professora Associada C/ Agregação, Universidade de Aveiro

vogais / examiners committee

Doutor Rui Jorge Morais Tomaz Valadas

Professor Catedrático, Universidade Técnica de Lisboa

Professor Doutor Paulo Jorge Salvador Serra Ferreira

Professor Auxiliar, Universidade de Aveiro

resumo

Uma entidade de internet é um denominador genérico de qualquer máquina (virtual / física) ou processo computacional no nó final ou no núcleo de uma rede, que gera ou recebe tráfego via Internet. A localização de entidades da Internet é vital para suportar uma grande quantidade de gestão, operação e segurança de serviços de rede. Nomeadamente, serviços inovadores no âmbito do IoT (Internet of Things) e / ou novos serviços de virtualização das funções da rede.

Esta dissertação implementa metodologias para resolver dois problemas. O primeiro é localizar entidades da Internet que estejam em comunicação com um ou mais servidores. O segundo problema é localizar uma entidade da Internet que esteja a infectar tabelas BGP e redirecionar tráfego através de ataques man-in-the-middle. A localização efectuada é ao nível físico (localização geográfica).

As metodologias de localização podem basear-se em medidas diretas e ativas ou fontes de dados indiretos (configurações de rede conhecidas e bancos de dados públicos disponíveis), no entanto, esta dissertação apenas baseia-se em medidas diretas entre a entidade de Internet e servidores, como por exemplo pings, não dependendo de terceiros para efectuar a localização.

Para conseguir a localização de uma entidade de internet, com as metodologias implementadas, é necessário conhecer a localização física dos servidores, bem como as medições de latência entre os servidores e a entidade de internet. Esta informação deve ser recolhida previamente.

abstract

An internet entity is a generic denominator of any machine (virtual/physical) or computational process at the end point or a core of a network, that generates or receives traffic via Internet. The localization of Internet entities is vital to support a multitude of management, operation and security of network services. Namely, novel services within the scope of the IoT (Internet of Things), and/or novel network functions virtualization services.

This dissertation implements methodologies to solve two problems. The first problem is to locate Internet entities that are exchanging traffic with one or more servers. The second problem is to locate an internet entity that is performing man-in-the-middle attacks on traffic redirection based on BGP route hijacking. The localization is at a physical level (geographic localization).

The localization methodologies may rely on direct and active measurements or indirect data sources (known network configurations and public available databases), however this dissertation only takes into account direct measurements, e.g. pings, between servers and the internet entity. This way it doesn't need to rely on third-parties to find the physical location.

To achieve the localization of an internet entity, with the implemented methodologies, it is necessary to know the physical location of the servers as well as the latency measurements between the servers and the internet entity. This information must be gathered beforehand.

Contents

List of Figures	I
List of Tables	III
Glossary.....	IV
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Structure.....	4
2 Context and State of the Art	5
2.1 Context.....	5
2.2 Methodologies	6
2.2.1. IP mapping-based.....	6
2.2.2. Measurement-based.....	6
2.3 Services	8
3 Localization Algorithm.....	10
3.1 Server Calibration	10
3.2 Host Localization	12
3.3 BGP MITM Attack Localization.....	16
3.4 Distance Between Coordinates.....	28
4 Geolocation Results	31
4.1 Physical location	32
4.2 World Visualization	34
4.3 Parsing Information and Data Errors	37
4.4 Test Servers	39
4.5 Results	40

4.5.1. Host Localization	40
4.5.1.1. Intercontinental Connections	41
4.5.1.2. Intracontinental connections.....	42
4.5.2. BGP MITM Attack Localization.....	43
4.5.2.1. Intercontinental connections.....	43
4.5.2.1. Intracontinental connections.....	44
4.6 Result Discussion.....	46
5 Conclusion.....	48
References.....	50

List of Figures

Figure 1.1 Example of a measurement-based technique ^[5]	3
Figure 2.1 Location estimation of a target host ^[26]	7
Figure 3.1 Circle with a Point and its Error Points	13
Figure 3.2 Drawn Circle, with its centre in Frankfurt and pinging London, on a World Map .	14
Figure 3.3 Distance between a point and two circles	15
Figure 3.4 Azimuth angle ^[41]	16
Figure 3.5 Ellipse ^[42]	17
Figure 3.6 Distance Between Servers	18
Figure 3.7 Distance Between two servers and an unknown host	18
Figure 3.8 Ellipse and b parameter	19
Figure 3.9 Ellipse and a parameter	20
Figure 3.10 Ellipse with Azimuth angles	22
Figure 3.11 Ellipse angle converted into Azimuth angle	23
Figure 3.12 Intersection of ellipses part 1	25
Figure 3.13 Intersection of ellipses part 2	25
Figure 3.14 Intersect Special Cases	27
Figure 3.15 Distance and Azimuth angle forming a new point	30
Figure 4.1 A 3-Dimensional Tree K-d Tree ^[38]	32
Figure 4.2 Resulting <i>k</i> -d tree for the point set (2,3), (5,4), (9, 6), (4, 7), (8,1), (7,2) ^[38]	32
Figure 4.3 World Map with an Ellipse drawn	34
Figure 4.4 The Robinson Projection with Tissot's indicatrix of deformation ^[47]	35

Figure 4.5 The Mercator Projection with Tissot's indicatrix of deformation ^[46]	36
Figure 5.1 A Quasi-Realistic Internet Graph ^[34]	49

List of Tables

Table 2.1 Available Database Fields ^[52]	8
Table 2.2 Geolocation Accuracy ^[52]	9
Table 3.1 Connection Ranking	11
Table 3.2 True north-based azimuth ^[41]	17
Table 4.1 File format	37
Table 4.2 Landmarks	40
Table 4.3 Host Localization problem results	46
Table 4.4 BGP MITM Localization problem results	47

Glossary

ISP	Internet Service Provider	GPS	Global Positioning System
IP	Internet Protocol	DNS	Domain Name System
IoT	Internet of Things	BGP	Border Gateway Protocol
RTT	Round Trip Time	TBG	Topology Based Geolocation
CBG	Constraint Based Geolocation	CAIDA	Center for Applied Internet Data Analysis
TSV	Tab Separated Values	WGS	World Geodetic System
MITM	Man-In-The-Middle	IERS	International Earth Rotation and Reference Systems Service
CSV	Comma Separated Values	ZIP	Zone Improvement Plan
API	Application Programming Interface		

1 Introduction

Geolocation^[1] is the identification or estimation of the real-world geographic location of an object, mobile phone, or Internet-connected computer terminal. In its simplest form geolocation involves the generation of a set of geographic coordinates and is closely related to the use of positioning systems, but its usefulness is enhanced by the use of these coordinates to determine a meaningful location, such as a street address. Determining a host's location on the Internet is vital to support a multitude of management, operation and security of network services.

This chapter is split into three sections. Firstly, the motivation of this work is explained. Second section contains the objectives of this thesis and, lastly, the structure of this work is described in section three.

The host institution of this thesis was the Aveiro Institute of Telecommunications.

1.1 Motivation

The concept of geolocation is often connected with **GPS** but there is more than one way to determine a host's location on the Internet. There are various methodologies that approach other solutions of localization and the jury is still out about which one is the best, each has its own strengths and weaknesses. These solutions only require an **IP** address and they'll either search on a public/private database or exploit information derived from **DNS**, network delay measurements, and inter-domain routing^[2]. All these methodologies were developed because locating traditional Internet users, such as users' desktop machines, is also relevant. They're mostly stationary and connected with a fixed wireline network, so it differs from the location methodologies used for mobile hosts.

The development of **IP** geolocation technology has a wide variety of usefulness, such as the ability to handle emergency services^[3], targeted news, advertisements and events (websites often tailor content, other than advertisements, based on geographic location), territorial rights management, etc. To achieve the necessity of all these services, the ability to locate someone must be independent from human user's information as much as possible because this information can be prone to errors. Particular clients may have incentive to evade geolocation. This includes cases in which privacy is important (e.g., for personal or humanitarian reasons) or related to either detecting or hiding illegal activity^[9]. These cases

may question the potential abuse of **IP** geolocation technology^[10], but this issue will not be discussed in this dissertation.

1.2 Objectives

This dissertation's main objective is the development, test and evaluation of novel methodologies for locating Internet hosts, Host Localization problem, and locate the entity behind a man-in-the-middle attack on traffic redirection based on **BGP** route hijacking^[31], the **BGP** MITM Attack Localization problem. For this purpose, the study and evaluation of the existing methodologies is an important part as well as their integration. There are two types of techniques for the Host Localization problem, namely, **IP** mapping-based techniques and measurement-based techniques.

The first type is the trickiest because one has to trust in third-party database. Private databases usually aren't free to access, and some countries even have their own database, e.g., Chinese **IP** Location databases^[4]. **ISP**'s also assign dynamically **IP**s for their clients, which make databases information easily out-dated. Examples of these techniques are *GeoTrack* and *GeoCluster* of the *IP2GEO*^[2] suite.

The second type relies on active interaction with the target system. They're basically conducted by pings and traceroutes. They assume that there is correlation between network latency and geographical distance, as shown in Figure 1.1. An example of these techniques is *GeoPing* of the *IP2GEO* ^[2] suite. These techniques are challenging because an **IP** address does not inherently contain an indication of location.

Figure 1.1 Example of a measurement-based technique [5]



There are no public techniques for the BGP MITM Attack Localization problem.

1.3 Structure

This dissertation is organized in 5 chapters. This chapter is the first and merely an introductory chapter that describes the motivation and objectives behind this work. In chapter 2, the highest level of development **IP** geolocation technology achieved to date is described and the services that use it. Chapter 3 explains the algorithm developed, chapter 4 how to find the location of the result coordinates and presents the results, and chapter 5 the conclusion.

2 Context and State of the Art

The interest in the geolocation technology is increasing. As a result, there has been much work on this area approaching the problem of locating hosts, and there are many commercially and publicly available products to solve the host location problem. However, most of these products require a fee. This chapter describes the best techniques developed and the services available on the Internet, with more focus on measurement-based methodologies. According to **CAIDA**^[33], the most accurate results are obtained by measurement-based approaches. This chapter will also give some context to the problems.

2.1 Context

As presented before, this dissertation aims to solve two problems. The first is locating an internet host and the second is to locate the entity behind a man-in-the-middle attack on traffic redirection based on **BGP** route hijacking. The **BGP MITM** Attack Localization problem is not in this chapter due to lack of content since there are no public methodologies available. That being the case, this section aims to give some context to the problem.

BGP is the routing protocol that allows world-wide Internet connectivity. This protocol inherent non-secure characteristics, together with the nonexistence of a trustable entity that correlates IP network prefixes with the autonomous-systems allowed to announce them, results in the vulnerability of the internet to multiple forms of attack or non-malicious misconfiguration. It then originated a problem because there were reports of Internet-scale traffic redirection based on **BGP** route hijacking, for perpetration of man-in-the-middle (at distance) attacks, that have put major institutions and network service providers in alert. Corporate customers have to content with a helpless bystander and victim roles due to the lack of tools to detect and counter-act Internet-scale traffic redirection^[31]. A platform that implements a method of detecting **BGP** routing attack without relying on **ISP**'s routing information and provides data in real time about these attacks was developed^[32]. The solution to this problem, takes this data into account and implements a new methodology based on the first problem methodologies.

2.2 Methodologies

Methodologies that approach **IP** Geolocation can be classified as **IP** mapping-based, or measurement-based. Some approaches make use of both methodologies and are defined as hybrids which is a sub-category of the measurement-based methodologies.

2.2.1. IP mapping-based

IP mapping-based techniques determine locations by retrieving information about a given **IP** address and matching queries against databases. Examples of such approaches are the **DNS LOC record**^[15], **Regional Internet Registry**^[16], by conducting **WHOIS**-lookups^[7], and Geolocation databases services offered by geoservices like *Maxmind*^[11], *db-ip*^[12], or *PlanetLab*^[13]. Specific implementations are *GeoCluster* and *GeoTrack* from the *IP2GEO*^[2]. *GeoCluster*^[2] couples partial host-to-location mapping information obtained (indirectly) from Websites with **BGP** routing information to infer the location of the host in interest. *GeoTrack*^[2] tries to infer location based on **DNS** names of the host of interest or other nearby nodes.

IP Mapping approach is used by web services like *ipinfo*^[8], *iplocation*^[6] and *ip2location*^[14], and network analysis tools that provide geolocation services like *whatroute*^[17] and *visualroute*^[18].

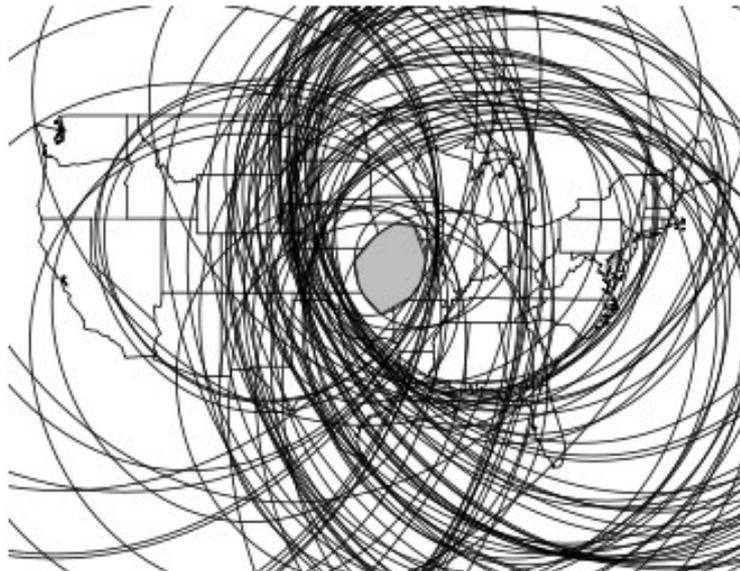
2.2.2. Measurement-based

These techniques rely on active interaction with the target system and they all assume that there is correlation between network latency and geographical distance. Shortest Ping^[25] is the simplest measurement-based technique possible: each target is mapped to the landmark that is closest to it in terms of measured **RTT**. The third technique of the *IP2GEO* suite, *GeoPing*^[2], uses network delay measurements made from geographically distributed locations to triangulate the coordinates of the host. It sets up latency vectors as a profile for the target and maps it to most similar landmark profile.

Furthermore, we can divide measurement-based techniques into constrained based, topology based and hybrid approaches. While hybrid approaches might be capable of integrating **IP** mapping-based strategies, they are still relying on active probing and thus considered measurement based.

Constrained Based Geolocation^[26] infers the geographic location of Internet Hosts using multilateration. Multilateration refers to the process of estimating a position using a sufficient number of distances to some fixed points, exhibited in Figure 2.1. As a result, it establishes a continuous space of answers instead of a discrete one.

Figure 2.1 Location estimation of a target host [26]



Topology Based Geolocation^[25] introduces topology measurements to simultaneously geolocate intermediate routers. Nonetheless, **TBG** is an enhanced version of the **CBG**. To overcome possible shortcomings and to verify the results hybrid techniques are developed. *Octant*^[27] is a framework for **IP** Geolocation and the current **State of the Art** in terms of active measurement-based approaches. The approximate host location is inferred by using geometric regions and thus a continuous solution space. By using *Bézier Curves*, large and complex areas can be represented in a precise way. *Octant* has a modular design, therefore is capable of using additional constraints like demographic data to improve accuracy. Other examples of hybrid approaches are *Posit*^[28], *HawkEyes*^[29] and *Spotter*^[30].

Almost all current measurement-based approaches are highly dependable on Landmarks. This creates a problem, the selection and placement of Landmarks out of a given set of possible locations for probing. *DRAGOON*^[5], the current **State of the Art** of solving the Landmark Problem, is an algorithm that finds optimized locations for Landmarks. Vantage Points are identified automatically in the topology in respect to minimize the maximum distance to the surrounding network topology.

2.3 Services

The best services of geolocation belong to **IP** Intelligence companies like *Akamai*^[19], *DigitalEnvoy*^[20], *Neustar*^[21], *F5*^[22], *Security Watchdog (Verifia)*^[23], *GeoPoint*^[24] *IPligence* ^[50], although this service is not the only service they provide. There are also a lot of web-sites like *ipinfo*^[8], *iplocation*^[6] and *ip2location*^[14], and API's like *eurekaAPI* ^[49] and even from google.

There is also network analysis tools that provide geolocation services like *whatroute*^[17] and *visualroute*^[18] and public databases like *Maxmind*^[11], *db-ip*^[12], *PlanetLab*^[13] or *ipinfodb* ^[51].

The following tables show results on comparison between some of the services.

	Neustar	MaxMind	IP2Location	IPligence
Continent	X	X	X	X
Country	X	X	X	X
State/Region	X	X	X	X
City	X	X	X	X
Zip Code	X	X	X	
Latitude	X	X	X	X
Longitude	X	X	X	X
ISP	X	X	X	X
Organization	X	X		
Organization Type	X	X		
ASN	X	X		
Net Speed	X	X	X	
Net Type	X	X		
Domain	X	X	X	
Area Code	X	X	X	
Metro Code	X	X	X	
DMA Code	X	X		
Weather Station			X	
Time Zone	X	X	X	X
Proxy	X	X	X	
Subscription	Query	Monthly	Annual	Annual

Table 2.1 Available Database Fields ^[52]

Distance	W3C	Neustar	MaxMind	IP2Location
0-2 km	50%	49%	33%	31%
2-10 km	9%	11%	9%	8%
10-25 km	4%	4%	6%	6%
25-50 km	6%	6%	6%	5%
50-100 km	5%	5%	8%	8%
100-250 km	3%	4%	7%	7%
250-500 km	3%	3%	6%	6%
500-1000 km	3%	3%	7%	8%
1000+ km	19%	14%	18%	20%

Table 2.2 Geolocation Accuracy ^[52]

3 Localization Algorithm

The localization algorithm implemented in this thesis aims to solve two problems. The first problem is locating an internet host based on a known latency measurement from servers that which the physical location is known. The second problem is locating an internet host that is redirecting **BGP** traffic from two servers communicating with each other. There are requirements to meet before the algorithm can proceed, them being: Server's physical location and the latency of the redirected, and the normal traffic between all the Servers. This information will affect the accuracy of this algorithm, which will be explained in detail further ahead in this chapter.

The result of the algorithm is an array of coordinates, which later will be used to find their physical location and give an estimative of where the internet host is, explained in chapter 4.

3.1 Server Calibration

Both problems convert a latency measurement into a **Meters per Milliseconds** measure, based on the distance between each other and the latency. Keeping in mind that a latency is a measure where a sender sends a packet to the receiver and then gets his response, therefore the average time between a server and another is half of the latency, the formula is:

$$DistanceLatency = \frac{2 \times DistanceBetweenServers}{LatencyBetweenServers}$$

Since there are many other factors that can influence a connection between two servers, this conversion may not be very accurate, specially at long distances. To solve this situation, an error threshold of distances is calculated. Usage of this error threshold is to intersect data by the problems, but this usage will be explained later, in this very chapter.

This error threshold calculation is done by:

1. Calibrate the servers independently

An average *DistanceLatency* is determined by ranking all the connections with a server and using the lowest rank. In the event of using a higher rank than 1, the program will warn the user that the accuracy will be affected. The formula used derives from the *DistanceLatency* formula:

$$AverageDistanceLatency = \frac{\sum DistanceBetweenServers}{\sum LatencyBetweenServers}$$

In case of rank 5, meaning there is no connections between this server and others, the formula calculates an average using all the connections provided, an average of the world.

Connection Rank	Connection Latency (MS)
1	< 100
2	< 250
3	< 500
4	> 500
5	None

Table 3.1 Connection Ranking

Calibrate Function

```

1. Initialize Total Meters to zero
2. Initialize Total Milliseconds to zero
3. Process (value):
4.   for every connection that has a given sender:
5.     if the connection has a maximum value lesser than(value):
6.       Add connection average value to Total Milliseconds
7.       Add distance between the sender and receiver of the connection
to Total Meters
8.   Do Process (100)
9.   if Total Milliseconds is zero:
10.    Do Process (250)
11.   if Total Milliseconds is zero:
12.    Do Process (500)
13.   if Total Milliseconds is zero:
14.    Do Process (∞):
15.     if Total Milliseconds is zero:
16.      for every connection that exists:
17.        Add connection average value to Total Milliseconds
18.        Add distance between the send and receiver of the
connection to Total Meters
19.   return Total Meters divided by Total Milliseconds
    
```

2. Compare *AverageDistanceLatency* to *DistanceLatency*

Each server has its own error but the *DistanceLatency* is always between two servers. By halving and then subtracting the *DistanceLatency* to the average, we deduce how much is deviating. The threshold is a percentage error of a given *DistanceLatency*. This being said, the formula for each server is:

$$ServerError = \frac{\left| AverageDistanceLatency - \frac{DistanceLatency}{2} \right|}{DistanceLatency}$$

Both *ServerErrors* are added and this value represents our error threshold.

1. C1 is the result Calibrate Function of Server1
2. C2 is the result Calibrate Function of Server2
3. Error1 equals the absolute value of the subtraction between C1 and (DistanceToLatency divided by two), divided by DistanceToLatency
4. Error2 equals the absolute value of the subtraction between C2 and (DistanceToLatency divided by two), divided by DistanceToLatency
5. **return** Error1 added **with** Error2

3.2 Host Localization

This problem aim is to locate an unknown host based only on latency measurements between the host and a server(s). The algorithm implemented can be divided in 3 steps, that will be detailed below: Firstly, the information is gathered and processed, explained in the next chapter, then a circle is draw based on the processed information and, lastly, the circles are intersected, and an array of physical coordinates will be the result. The more information given, the more accurate this algorithm will be.

➤ First Step: Drawing a Circle

When the process of gathering information and calibrating a server is done, the requirements, to solve the Host Localization, are met. Each connection between a server and the unknown host provided will have a circle drawn around the server, requiring a few steps.

- Calibrating a server and determining its error

The first step is calculating the *DistanceLatency* and *ServerError* by following the Chapter 3.1 methods of all servers with connection to the unknown host.

- Determine the Radius

Given that we have the attributes of a ping between a server and a host, we use the previously calculated *DistanceLatency*

$$Radius = DistanceLatency \times AveragePing$$

Circle data Function

1. DistanceToLatency is the result of Calibrate a given Server
2. **return** circle object **with** Server coordinates, DistanceToLatency multiplied by average ping and the Error

- Draw Circle Points

With its centre on a server, 72 points are drawn, one point every 5 degrees, from 0 to 360 degrees. To draw these points, given a coordinate (latitude, longitude), an angle and a distance, the algorithm determines the final position (This step will be discussed further ahead in section 3.4). The *SampleError* is also used to draw 10 extra points, attached to a point. This being said, a drawn point consists of a latitude, longitude and its error points (which are coordinates). These 10 error points consist on 5 points drawn towards the centre and 5 points drawn away from the centre, as in figure 3.1.

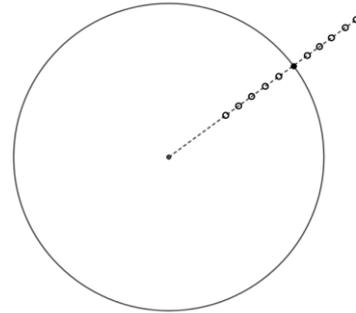


Figure 3.1 Circle with a Point and its Error Points

Their distance calculation follows the formulas,

$$ErrorDistance = Radius \times SampleError$$

$$ErrorPointDistance = \frac{Radius \pm ErrorDistance \times N}{Max(N)}$$

N is a number that goes from 5 to 1, representing the error points ($N = 5$ is the first error point, $N = 4$ the second, etc).

Draw Circles Function

```

1. Initialize angle to zero
2. Initialize point array
3. while the angle is different from 360:
4.   Initialize error array
5.   temporary point equals result of New Coordinates function (Circle
   center, angle, Radius)
6.   while X is 5 and is between 5 and 1, subtracting 1 every iteration:
7.     Error distance is Radius multiplied by the Error
8.     Add to Error array the result of New Coordinates function (Circle
   center, angle, Radius added to Error Distance Multiplied by X, all divided
   by 5)
9.     Add to Error array the result of New Coordinates function (Circle
   center, angle, Radius subtracted to Error Distance Multiplied by X, all
   divided by 5)
10.  Add to point array an object with the temporary point and the Error
   Array
11.  Add five to the angle
12.  return point array

```

All these steps are repeated for every connection provided to find the unknown host, creating an array of circles. The final result will be the figure 3.2, with the error points hidden.

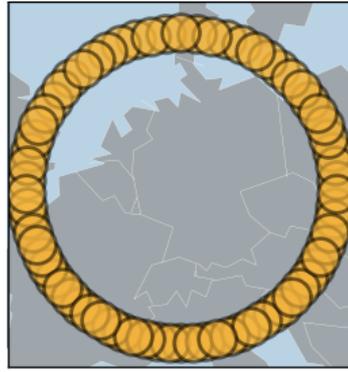


Figure 3.2 Drawn Circle, with its centre in Frankfurt and pinging London, on a World Map

➤ Second Step: Intersection

The intersection between circles is the last step to get the final coordinates. If there is only one circle drawn, the algorithm will return all the points of that circle. In-case of the number of existing circles is more than two, the algorithm does the following steps:

- Find coordinates

From each circle, two points are chosen that meet the following criteria: using the formula,

$$PointDistanceToCircles = \sum Min(DistanceToOtherCircles)$$

The two points, that have the lowest *PointDistanceToCircles* (the sum of all distances between a point and other drawn circles) value, are chosen. The *DistanceToOtherCircles* also takes into account the error points. This distance consists on finding the minimal value between a point and a circle, comparing a point to all points, and respective error points, drawn of a circle and determine the minimal value, as shown in figure 3.3.

Distance to Circles Function

```

1. Initialize final distance to zero
2. for each drawn circle:
3.   Initialize auxiliary distance to -1
4.   for each point, p, in each drawn circle:
5.     Initialize distance equal to Distance between a point given and p
6.     if auxiliary distance is -1:
7.       auxiliary distance equals distance
8.     else if auxiliary distance is greater than distance:
9.       auxiliary distance equals distance
10.    for each error point in p:
11.      distance equals to Distance between a point given and error
point
12.      if auxiliary distance is greater than distance:
13.        auxiliary distance equals distance
14.    if auxiliary distance is different than -1:
15.      Add auxiliary distance to final distance
16. return final distance

```

Not all error points are taken into account. The algorithm decides which 5 points will be used in comparison, either the 5 closest to the centre or the 5 further away from the centre. After deciding which 5 to use, if a distance to an error point is bigger than a distance to a previous error point or to the normal point, the algorithm will not make further comparisons.

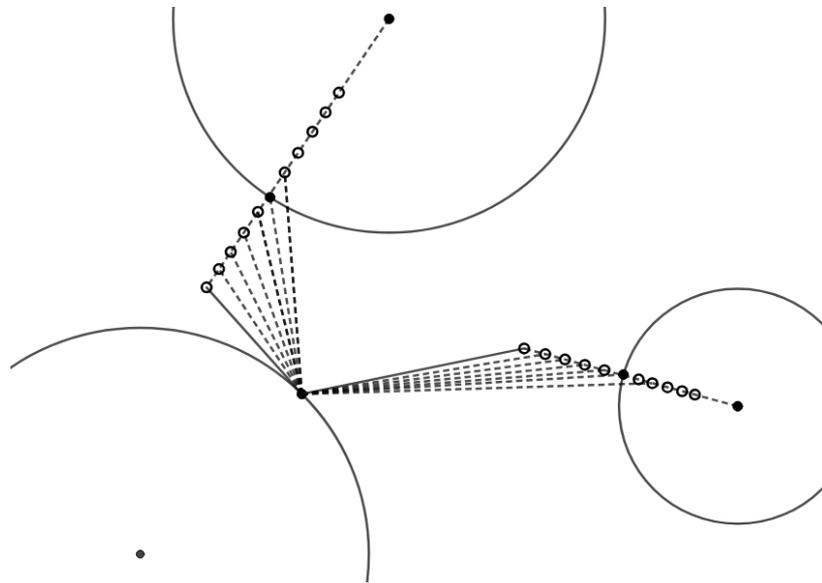


Figure 3.3 Distance between a point and two circles

- Process the error

After choosing two points, it follows the same process but comparing only the error points of these chosen points. If it finds smaller distances, those error points will be the final coordinates. The algorithm will then proceed to find these coordinates location, described in detail in the next chapter.

1. Initialize restrictions
2. Initialize points
3. **while** there is information:
4. Draw Circle Data and add it to restrictions
5. Initialize grid
6. **while** there are restrictions:
7. Draw Circle based on restrictions and add it to grid
8. **if** length of the grid is one:
9. points equals to grid points
10. **else:**
11. **for** each circle in the grid:
12. Get two points with the minimum distance using Distance To Circles Function (point)
13. **if** the point errors of those two points have smaller distance using the same function:
14. Replace them
15. Append the two points to points

3.3 BGP MITM Attack Localization

This problem aim is to locate an internet host that is redirecting **BGP** traffic from two servers communicating with each other. To solve this, the algorithm draws ellipses or circles according to the information given. This problem steps are similar to those of the first problem: Firstly, the information is gathered and processed, explained in chapter 4, then an ellipse or a circle is draw based on the processed information and, lastly, the ellipses and/or circles are intersected, and an array of physical coordinates will be the result. In addition to draw ellipses, the way they're intersected to other ellipses/circles is different.

➤ First Step: Drawing an Ellipse

This problem is much more complex than the previous one. After gathering information, between each connection provided and determining the *DistanceLatency* and *ServerError*, an ellipse will be drawn. This draw consists in various steps.

- Determine information between servers

Given two server coordinates, the algorithm will determine a middle point between them (More detail about this part later on, in section 3.4). This middle point will be the centre point of the ellipse. After knowing the middle point, the algorithm calculates the angle between a server coordinate and the middle point. This angle

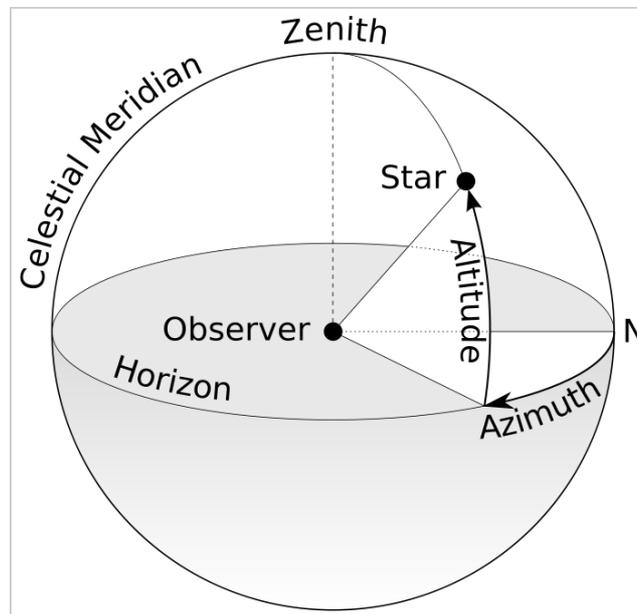


Figure 3.4 Azimuth angle ^[41]

has a special angular measurement called azimuth ^[41]. This angle is measured in degrees and is the angle formed between a referenced direction (North, for example) and a line from the observer to a point of interest.

The algorithm uses a north-based Azimuth with the following values:

North	0°	South	180°
North-northeast	22.5°	South-southwest	202.5°
Northeast	45°	Southwest	225°
East-northeast	67.5°	West-southwest	247.5°
East	90°	West	270°
East-southeast	112.5°	West-northwest	292.5°
Southeast	135°	Northwest	315°
South-southeast	157.5°	North-northwest	337.5°

Table 3.2 True north-based azimuth [41]

- Defining an ellipse

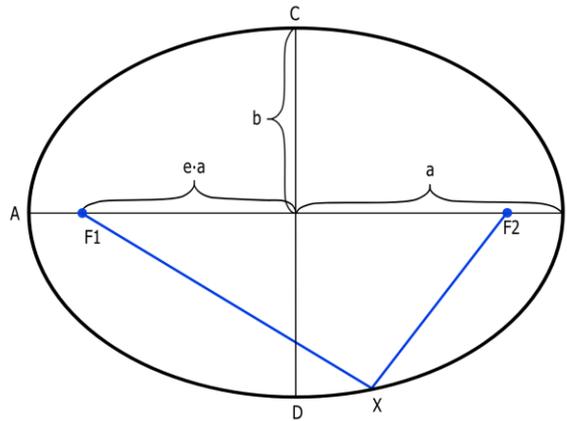


Figure 3.5 Ellipse [42]

An ellipse is composed by its centre and two parameters, a b and an a , as shown in figure 3.5. Being *NormalPing* the ping between two servers that are not don't have their packets relayed, and *RelayedPing* the ping between two servers that have their packets relayed. The distance between the two servers is X_d (Figure 3.6). Since the data only shows a latency measurement obtained by a ping that consists of sending packets and receiving the response, meaning a packet goes to the receiver and comes back traveling twice the distance, X_d can be determined using the following formula,

$$NormalPing \times DistanceLatency = 2 \times X_d \Leftrightarrow$$

$$X_d = \frac{NormalPing}{2} \times DistanceLatency$$

$$X = \frac{NormalPing}{2}$$

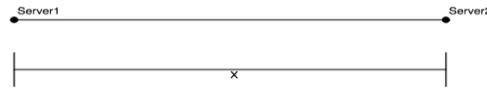


Figure 3.6 Distance Between Servers

The distance between a server and an unknown host is $Y1_d$ and the other server and the same unknown host is $Y2_d$ (Figure 3.7). This unknown host that relays packets can be anywhere and the total distance between them is represented as,

$$RelayedPing \times DistanceLatency = X_d + Y1_d + Y2_d$$

$$RelayedPing = X + Y1 + Y2$$

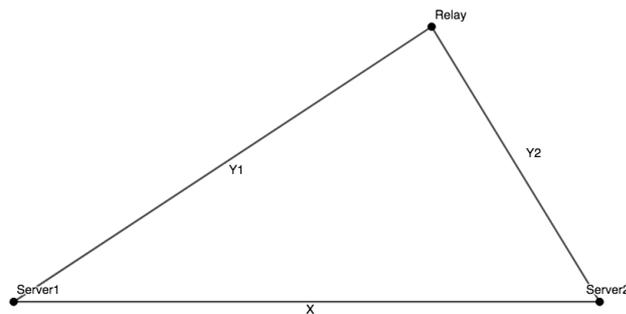


Figure 3.7 Distance Between two servers and an unknown host

To determine the a and b parameters two situations are taken into account. These situations are specific cases of where the unknown host is. To make the calculations easier, only after the algorithm calculates the two parameters it takes *DistanceLatency* into account.

- b parameter

To determine the b parameter, the algorithm uses the situation where the host is directly above the centre of the ellipse (figure x).

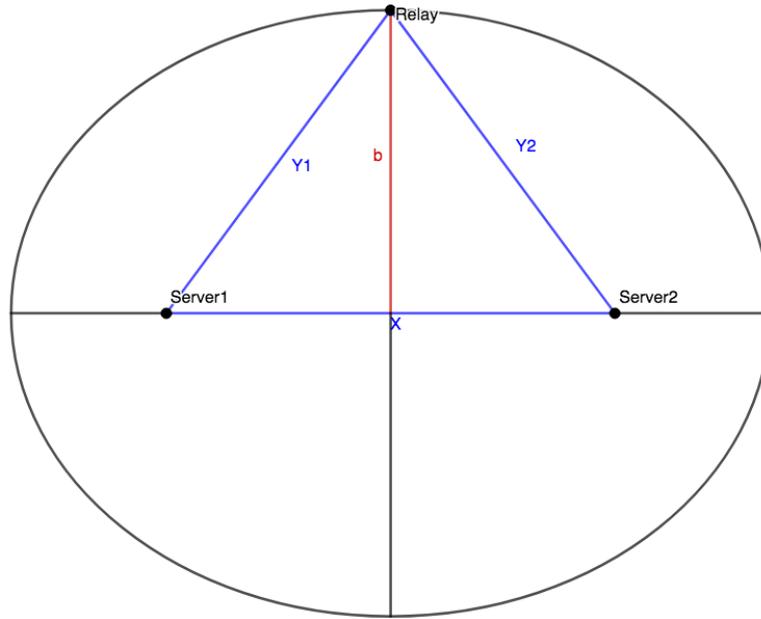


Figure 3.8 Ellipse and b parameter

In this situation, the distance between the two servers and the unknown host is the same, meaning,

$$Y1 = Y2$$

Applying this formula to the previous ones,

$$RelayedPing = X + Y1 + Y2 \Leftrightarrow$$

$$RelayedPing = X + 2 \times Y1 \Leftrightarrow$$

$$Y1 = \frac{RelayedPing - X}{2}$$

After $Y1$ is known, the *DistanceLatency* is taken into account,

$$Y1_d = Y1 \times DistanceLatency$$

$$X_d = X \times DistanceLatency$$

and, since the triangle formed is rectangular, the Pythagorean theorem is used to calculate the b parameter. This theorem states the square of the hypotenuse (the side opposite to the right angle) is equal to the sum of the squares of the other two sides [43].

$$a^2 + b^2 = c^2$$

In this case $Y1_d$ is the hypotenuse and one of the sides is half of the distance between two servers, therefore,

$$Y1_d^2 = \left(\frac{X_d}{2}\right)^2 + b^2 \Leftrightarrow$$

$$b = \sqrt{Y1_d^2 - \left(\frac{X_d}{2}\right)^2}$$

b parameter is a distance. It cannot be negative.

- a parameter

To determine the a parameter, the algorithm uses the situation where the host is to the left of the *Server1* (Figure 3.9).

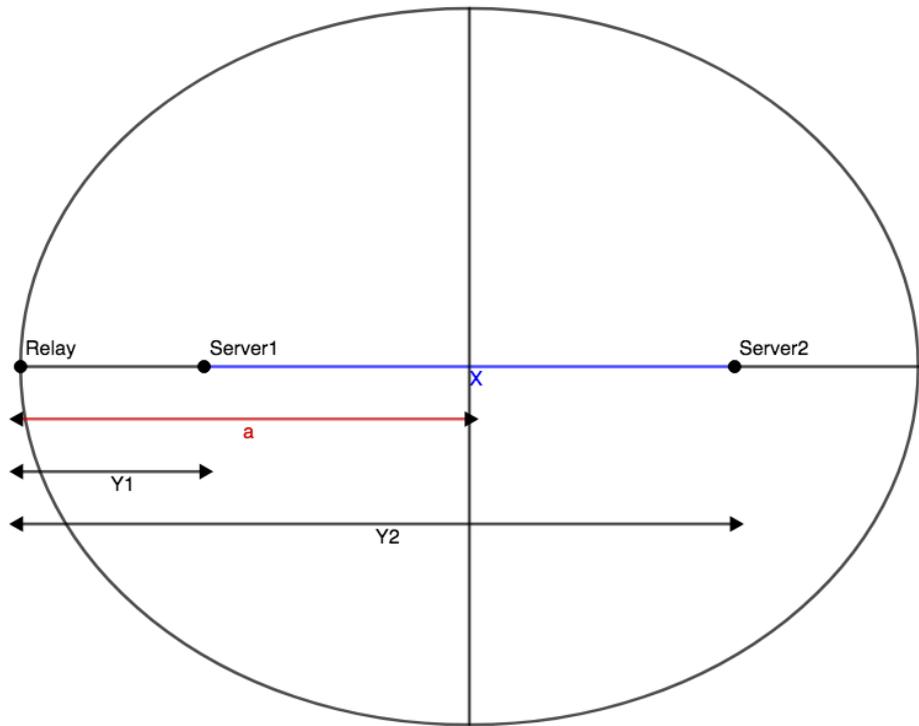


Figure 3.9 Ellipse and a parameter

The conclusion of this situation is the following,

$$\begin{cases} Y2 = Y1 + X \\ a = Y1_d + \frac{X_d}{2} \end{cases}$$

Since X_d is already known, $Y1_d$ is calculated applying the previous conclusion to the other formulas,

$$RelayedPing = X + Y1 + Y2 \Leftrightarrow$$

$$RelayedPing = 2 \times (X + Y1) \Leftrightarrow$$

$$Y1 = \frac{RelayedPing}{2} - X$$

Taking *DistanceLatency* into account,

$$Y1_d = Y1 \times DistanceLatency$$

a parameter is then calculated.

Draw Ellipse Data Function

1. Get distance and angle between the two landmarks using Distance Between Points function
2. Midpoint equals result of New Coordinates Function (landmark point, angle, distance divided by two)
3. X equals to normal ping average divided by two
4. $DistanceToLatency$ equals distance divided by X
5. $Y1$ equals to (ping relayed average divided by two, subtracting X) multiplied by $DistanceToLatency$
6. 'a' parameter equals to distance divided by two, adding $Y2$
7. $Y2$ equals to ((ping relayed average subtracted by X) divided by two) multiplied by $DistanceToLatency$
8. 'b' parameter equals to square root of ($y1$ squared subtracted by (distance divided by two) squared)
9. Error equals to Error Calculation Function added by normal ping error and relayed ping error
10. **return** ellipse object with both landmarks, midpoint, 'a', 'b', error

- Drawing the ellipse

Having the a and b parameters, the ellipse can now be drawn. The algorithm draws the four main points first, point A, B, C and D (Figure 3.5).

Point A is calculated from the middle point adding the a parameter towards the azimuth angle, between the *Server2* and the middle point.

Point B is calculated from the middle point adding the a parameter towards the azimuth angle, between the *Server1* and the middle point.

Point C is calculated from the middle point adding the b parameter towards the azimuth angle, between the middle point and the *Server2*, minus 90 degrees (Figure 3.10).

Point D is calculated from the middle point adding the b parameter towards the azimuth angle, between the middle point and the *Server2*, adding 90 degrees (Figure 3.10).

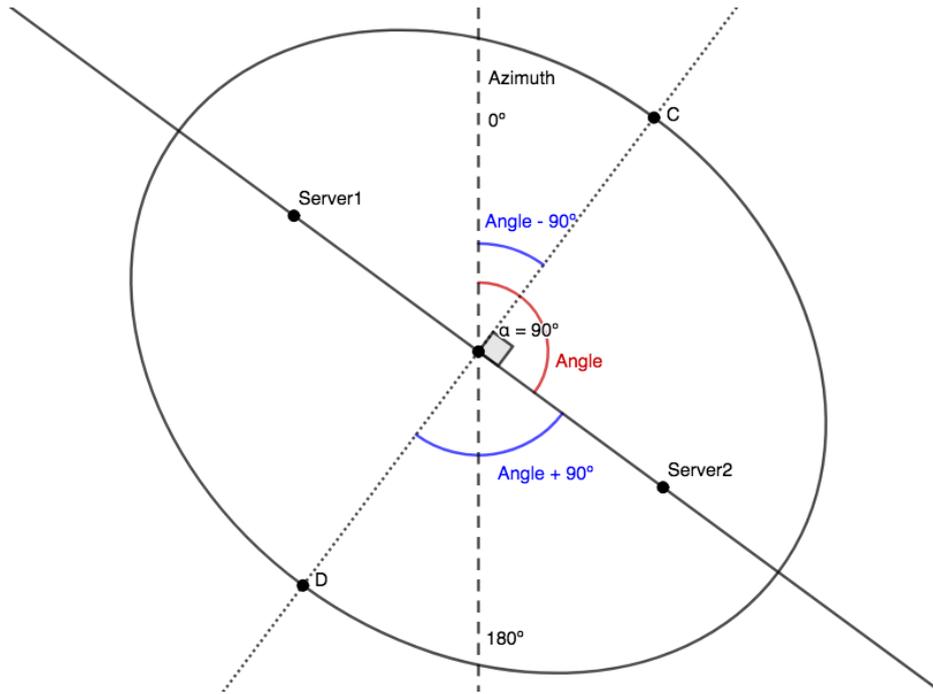


Figure 3.10 Ellipse with Azimuth angles

After drawing the four main points, the *TotalError* is calculated using the following formula,

$$TotalError = NormalSampleError + RelayedSampleError + ServerError$$

Being the *NormalSampleError* the *SampleError* of the connections that are not relayed, the *RelayedSampleError* the *SampleError* of the connections that are relayed and the *ServerError* the error from the calibration of the servers.

With its centre point on the middle point located between *Server1* and *Server2*, 360 points are drawn, one point every degree, from 0 to 360 degrees, making an ellipse. To draw these points, given the middle point as a coordinate (latitude, longitude), an angle, β , and a distance, d , the algorithm determines the final position (This step will be discussed further ahead in section 3.5). The distance, d , given is calculated using the formula of the distance between the centre of an ellipse and a point, this formula consists of giving the parameter a, b and a certain angle θ converted into radians,

$$x = a \times \cos \theta_{radians}$$

$$y = b \times \sin \theta_{radians}$$

$$d = \sqrt{x^2 + y^2}$$

d is a distance. It cannot be negative.

The angle θ is then converted into azimuth angle (Figure 3.11) using the following formula,

$$\beta = \lambda - \theta$$

Being λ the azimuth angle previously calculated between *Server1* and the middle point and β the azimuth angle needed to draw the point.

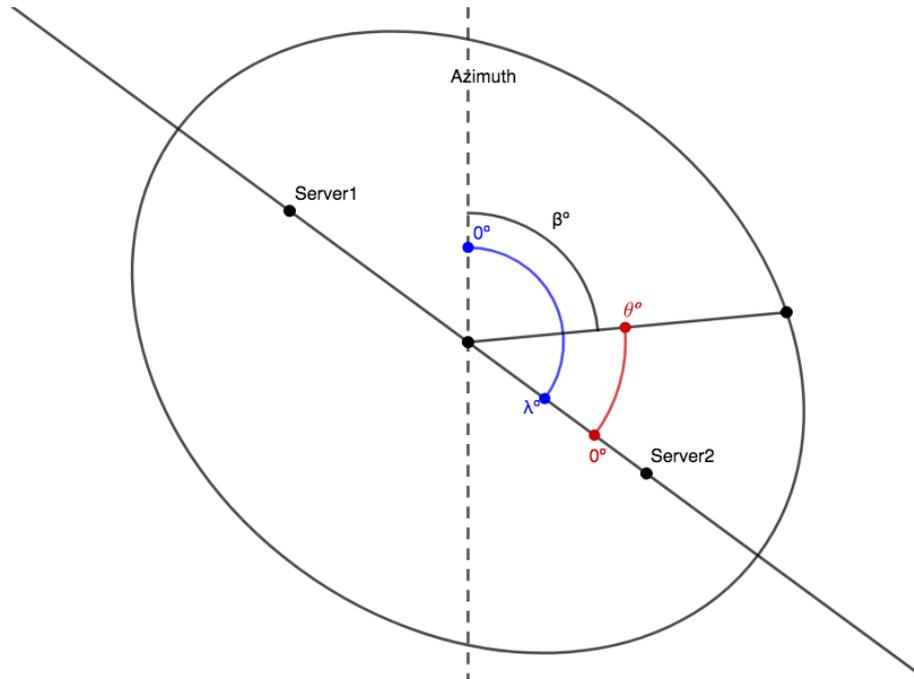


Figure 3.11 Ellipse angle converted into Azimuth angle

The *TotalError* is also used to draw 10 extra points, attached to a point. This being said, a drawn point consists of a latitude, longitude and its error points (which are coordinates). These 10 error points consist on 5 points drawn towards the centre and 5 points drawn away from the centre, just like the previous problem.

Their distance calculation follows the formulas,

$$ErrorDistance = d \times TotalError$$

$$ErrorPointDistance = \frac{d \pm ErrorDistance \times N}{Max(N)}$$

N is a number that goes from 5 to 1, representing the error points ($N = 5$ is the first error point, $N = 4$ the second, etc).

All these steps are repeated for every connection provided to find the unknown host, creating an array of ellipses.

Draw Ellipse Function

```

1. Initialize angle to zero
2. Initialize point array
3. while angle is not 360:
4.     Initialize error array:
5.     Distance equals square root of ((‘a’ parameter multiplied by cosine of
angle in radians) squared added to (‘b’ parameter multiplied by sine of angle in
radians) squared)
6.     Final angle equals to azimuth1 - angle
7.     Temporary point equals to New Coordinates Function (midpoint, final angle,
distance)
8.     while X is 5 and is between 5 and 1, subtracting 1 every iteration:
9.         Distance error equals to distance multiplied by data error
10.        Draw two error points and add them to error array
11.        Add temporary point and its errors to point array
12.        Add 1 to the angle
13. return point array

```

➤ **First Step: Drawing a Circle**

In this problem, drawing a circle is a rare occasion but it is needed. Whenever there’s information of a connection between a server and itself, and it is being relayed, a circle is drawn. The way this problem draws a circle is exactly the same way the Host Localization (Chapter 3.3.1) draws. The only difference is in the error calculation. Instead of using the *SampleError* it uses the *TotalError* calculated using the following formula,

$$TotalError = NormalSampleError + RelayedSampleError$$

Being the *NormalSampleError* the *SampleError* of the connections that are not relayed and the *RelayedSampleError* the *SampleError* of the connections that are relayed.

```

1. DistanceToLatency is the result of Calibrate a given Server
2. return Circle object with Server coordinates, DistanceToLatency multiplied
by average ping and the Normal Error added to the Relayed Error

```

➤ **Second Step: Intersection**

The process to intersect the ellipses/circles drawn with each other has a few steps. The algorithm analyses one ellipse/circle at a time and compares it with the others.

- Intersection between ellipses

The first ellipse in the array is the first one that will be compared to the others. Every drawn point will be tested to see if this point intersects with every other ellipse drawn. To accomplish this, the algorithm determines the angle, θ , and distance, d , between a point, P , and the middle point of the other ellipse (Figure 3.12).

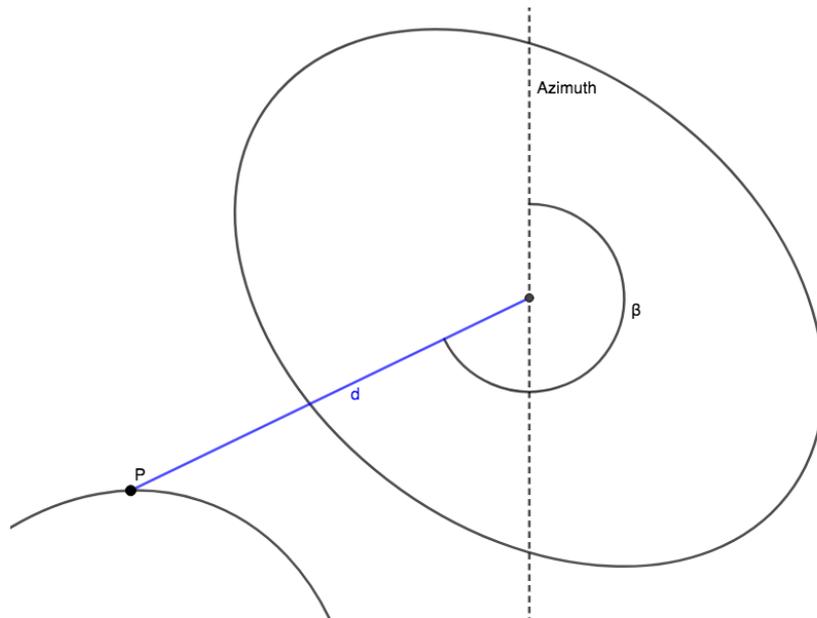


Figure 3.12 Intersection of ellipses part 1

This angle is then converted into an ellipse angle, the reverse process of figure 3.11, using the formula,

$$\theta = \lambda - \beta$$

and used to calculate the distance, k , between the middle point and an ellipse point with that angle, θ (Figure).

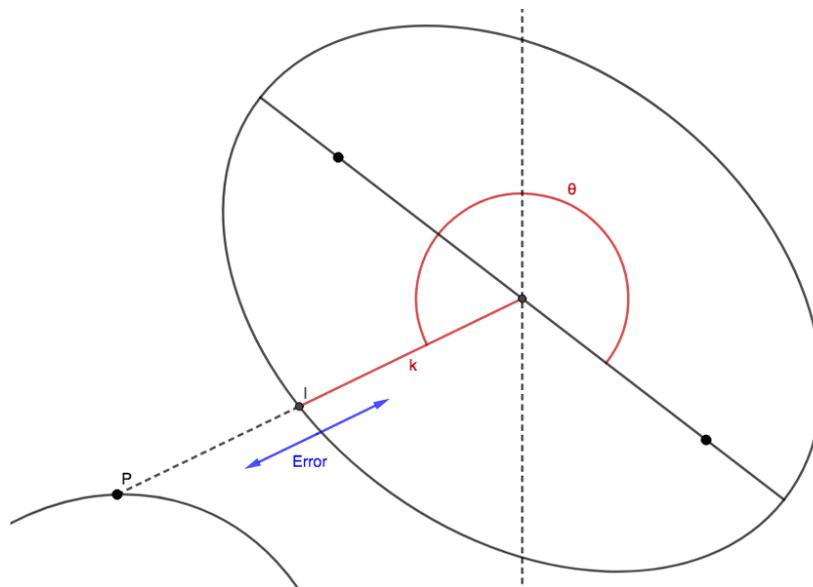


Figure 3.13 Intersection of ellipses part 2

After having both distances, an *ErrorDistance* is calculated their values are compared.

$$ErrorDistance = k \times TotalError$$

The point will pass the test if the distance d is between the error interval of the distance k ,

$$k - ErrorDistance \leq d \leq k + ErrorDistance$$

This process is repeated to every other ellipse and a point will only pass the test if it meets the previous condition to every comparison. If an ellipse doesn't have intersected points, it will be removed, and this process will restart again without it. This process will be detailed further ahead as a Special Case.

Find Point Ellipse Function

1. Distance and angle are the result of Distance Between Points function (midpoint, a point)
2. Auxiliary distance equals result of Ellipse Point function ('a', 'b', angle subtracted by azimuth1)
3. Error distance equals auxiliary distance multiplied by error
4. **if** distance is between auxiliary distance subtracted by error distance and auxiliary distance added with error distance:
5. **return** True
6. **return** False

- Intersection to a Circle

In case of comparing to a point to a circle, there is no need of angle calculation or the calculation of the distance d . Since the distance between any point of any angle is the same,

$$d = Radius$$

A circle will also be removed if it doesn't have any intersected points.

Find Point Circle Function

1. Distance equals Distance Between Points function (two points)
2. Error distance equals distance multiplied by error
3. **if** radius is between distance subtracted by error distance and distance added with error distance:
4. **return** True
5. **return** False

When all the points of an ellipse/circle have gone through this process, the intersected points, if there are any, join an array of points that is the result of this algorithm. These points are then needed to find their physical location.

Find Intersection Function

1. Initialize find to True
2. **for** each circle or ellipse in grid:
3. find equals result of Find Point Ellipse/Circle function (point)
4. **if** find is True:
5. **return** point
6. **else:**
7. **for** each error in point:
8. Initialize find to True
9. Find equals result of Find Point Ellipse/Circle function (error)
10. **if** find is True:
11. **return** point
12. **return** none

➤ Third Step: Special Intersection Cases

```

Search Function
1. Initialize restart to False
2. Initialize count to 0
3. Initialize temporary grid
4. Initialize deleted grid
5. for each object in grid:
6.     Initialize find to False
7.     Initialize temporary points
8.     for each point in object:
9.         Points found equals to result of Find Intersection function (point,
grid)
10.        if points found is not none:
11.            Add points found to temporary points
12.            Find equals to True
13.        if find equals to True:
14.            Add Object with temporary points to temporary grid
15.        else:
16.            Restart equals True
17.            Break the loop
18.        Add 1 to count
19.    if restart is True
20.        Add to deleted grid object number (count) of the grid
21.    Temporary grid and auxiliary deleted grid equals to result of Search
Function(grid)
22.    if deleted grid is not None:
23.        Add to deleted grid the auxiliary deleted grid
24.    return temporary grid and deleted grid

```

After removing an ellipse/circle and restarting the previous process, these objects will go to a new array. When the previous process is completed, the algorithm will try to intersect any object in the created array to the ellipses/circles that have intersected points (e.g. Figure 3.14). In case of failure, the algorithm ignores the objects in the array. In case of success, the algorithm has two choices:

- If there's only one successful object intersected, the final result will add this object's result and put together all the intersected points.
- If there's more than one successful object, the algorithm will ignore them all since there's no way to know which one is the real one.

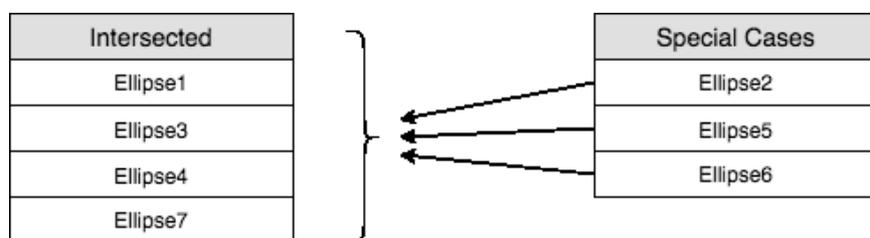


Figure 3.14 Intersect Special Cases

```

1. Initialize restrictions
2. while there is information:
3.     Append Circle or Ellipse Data Function to restrictions
4. Initialize grid
5. Initialize deleted
6. for each restriction:
7.     Append Circle or Ellipse object result of Draw Ellipse/Circle Function
8. Grid and deleted equals the result of Search Function (grid)
9. Initialize success deleted
10. Initialize success grid
11. for each object in deleted:
12.     Temporary grid clones grid
13.     Add deleted to temporary grid
14.     Temporary grid and temporary deleted equals result of Search Function
(temporary grid)
15.     if temporary deleted is not None:
16.         Add object to success deleted
17.         Add temporary grid to success grid
18. if success grid has only one object:
19.     Grid equals success grid

```

3.4 Distance Between Coordinates

The system used to find distances between coordinates is the same used by the **GPS**. This system is called **WGS** and the version used is the **WGS 84**, which is the last version. This system is a standard for use in cartography, geodesy, and satellite navigation ^[44]. In this system the Earth's center mass is the origin coordinate and the meridian of zero longitude is the IERS Reference Meridian, 5.3 arc seconds east of the Greenwich meridian at the latitude of the Royal Observatory ^[44]. Currently, WGS 94 uses the Earth Gravitational Model 1996 geoid, revised in 2004. This geoid defines the normal sea level surface by means of a spherical harmonics series of degree 360 ^[44]. **WGS 84** uses radio waves transmitted by satellites to create ellipsoid models. The error of this system is believed to be less than 2cm.

Distance Between Points Function

```

1. Distance and angle equals Geodesic Inverse Function (point1 latitude, point1
longitude, point2 latitude, point2 longitude)
2. return distance and angle

```

The module used is Geodesic from Geographiclib ^[45]. This module has a lot of functions, but the algorithm only uses two of them. Those are:

- Direct()

This function solves the direct geodesic problem, which is given a first point coordinate (latitude and longitude), an azimuth angle and a distance, it returns a second point coordinate (Figure 3.15).

Parameters are:

- Latitude of the first point in degrees
- Longitude of the first point in degrees
- Azimuth at the first point in degrees
- Distance between the first point and second point in meters

Function returns:

- Latitude of the first point in degrees
 - Longitude of the first point in degrees
 - Azimuth angle between the first point and second point in degrees
 - Latitude of the second point in degrees
 - Longitude of the second point in degrees
 - Azimuth angle between the second point and the first point in degrees
 - Distance between the two points in meters
 - Spherical arc length from the first point to the second in degrees
- Inverse()

This function solves the inverse geodesic problem, which is given two points coordinates, it returns the azimuth angles and the distance between them.

Parameters are:

- Latitude of the first point in degrees
- Longitude of the first point in degrees
- Latitude of the second point in degrees
- Longitude of the second point in degrees

Function returns:

- Latitude of the first point in degrees
- Longitude of the first point in degrees
- Azimuth angle between the first point and the second point in degrees
- Latitude of the second point in degrees
- Longitude of the second point in degrees
- Azimuth angle between the second point and the first point in degrees
- Distance between the first point and second point in meters
- Spherical arch length from the first point to the second in degrees

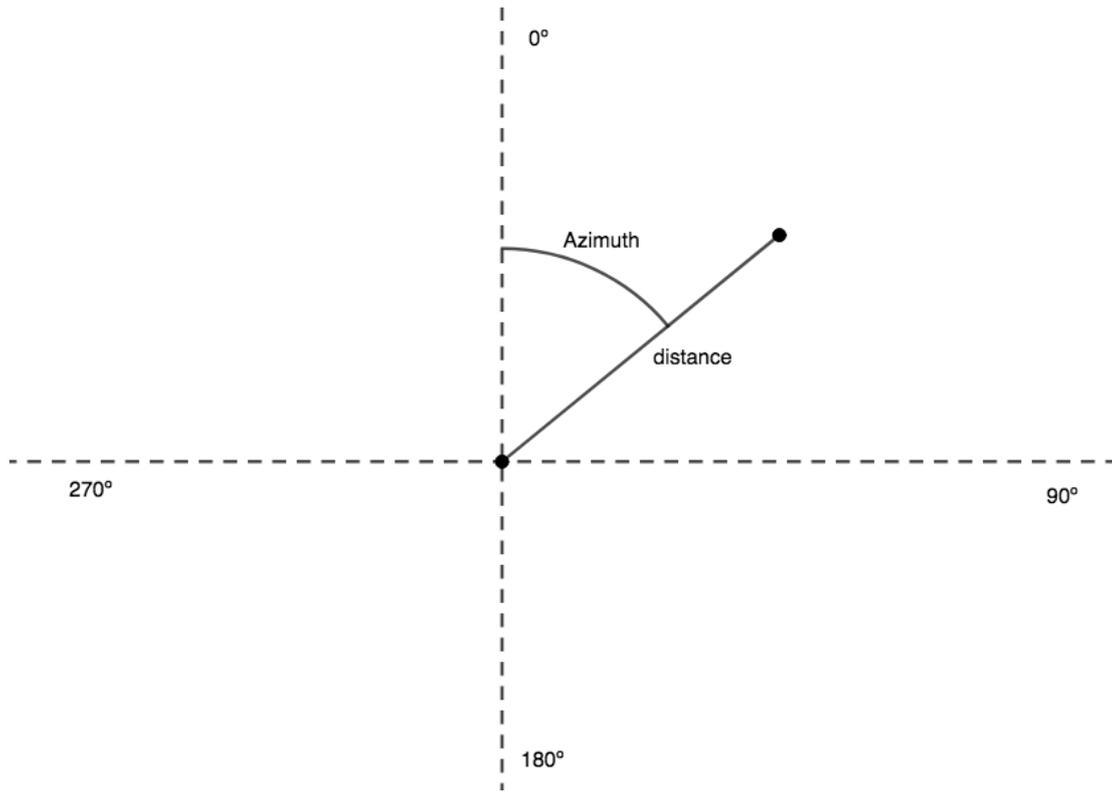


Figure 3.15 Distance and Azimuth angle forming a new point

New Coordinates Function

1. Final Point equals Geodesic Direct Function (point latitude, point longitude, angle, distance)
2. **return** final point

4 Geolocalization Results

Geolocalization is the attachment of a geolocation to an object, the identification of a geographic location of an unknown host, which is this chapter's goal. The process used to achieve this goal is called reverse geocoding. Reverse geocoding consists of back coding a point location to a readable address or place name ^[48]. This allows the identification of countries, cities, states, counties, street addresses and **ZIP** codes, which are easier to understand by the end user.

The reverse geocoding algorithm, on how the physical location is found, consists of receiving an array of coordinates, composed by latitude and longitude, not displaying their errors, that are the result of the previously described algorithm in chapter 3 and transform them into an array of country names. The process used does not require any Internet connection, although other ways to approach this problem do, and is the final step of the program. The algorithm to find the country of a coordinate uses a set of known locations to build a k -dimensional tree and search the nearest point in the tree of the input coordinate, resulting in an array of country names.

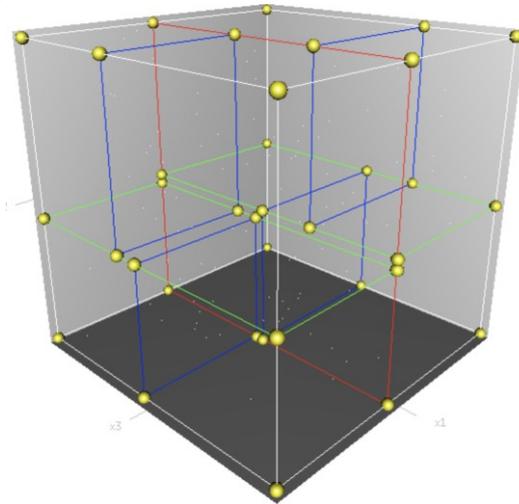
After discovering the country name of all given coordinates, this process also draws two two-dimensional maps, a world map and a close-up map with all the coordinates given by the previous procedure. This procedure is merely to add some context to the final coordinates since a name of a country can be abstract when wanting to know a location, specially because the algorithm to find a coordinate physical location always returns a country name, even if the coordinate is in the middle of an ocean.

Also, in this chapter, the final results of the program are displayed and discussed. To test the algorithm, the two problems require normal and relayed ping information. To meet this requirement, both these problems need two folders to work with. One with non-relayed

latency measurements between all servers, in **tsv** format, and another with latency measurements from a server to an unknown host, in case of the Host Localization problem, or relayed latency measurements from a server to another server, in case of the **BGP MITM Attack Localization** problem, also in **tsv** format.

4.1 Physical location

Finding the location of an array of coordinates is the final step of the program. To remove any internet restrictions and dependence towards web **API**'s, like Google Reverse Geocoding, a module that works off-line was implemented. This module has a set of known geocoded locations, various coordinates associated with each country inside a **.csv** file, builds a k -dimensional tree with the locations known and uses a Nearest Neighbour Search to find the closest neighbour of a given coordinate. This type of tree is a binary tree in which every node is a k -dimensional point. Every non-leaf node recursively divides space into two parts, generating a splitting hyperplane (a subspace whose dimension is one less than that of the k -dimensional tree, **Figure 4.1 A 3-Dimensional Tree K-d Tree** ^[38]



e.g.: a 3-dimensional would split into various 2-dimensional hyperplanes, as seen in Figure 4.1). That node's left tree is constituted by points to the left of this hyperplane, and the right tree is constituted by points to the right of the hyperplane. Example of a final k -d tree in figure 4.2.

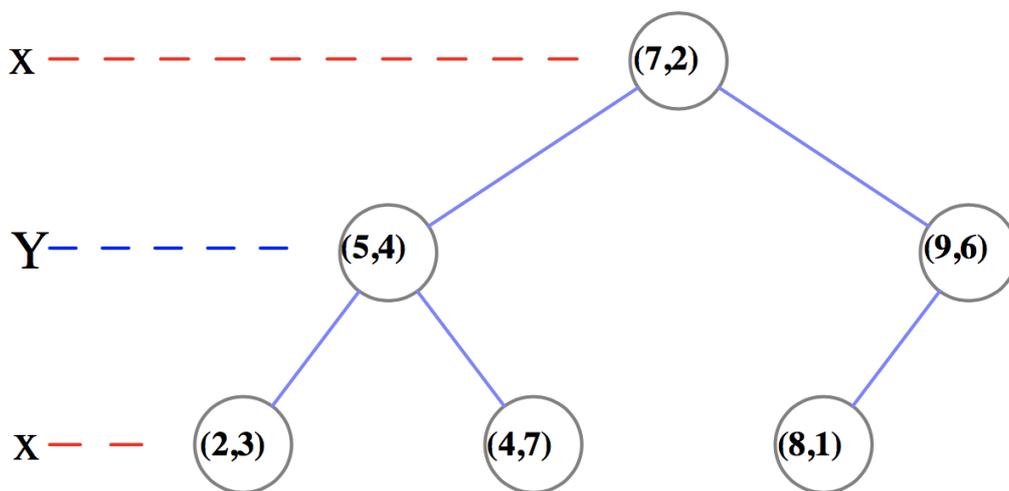


Figure 4.2 Resulting k -d tree for the point set $(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)$ ^[38]

It then proceeds to use the k -nearest neighbour search to a given coordinate which identifies the top k nearest neighbours. This search is done efficiently by using the trees properties to quickly eliminate large portions of the search space. Searching for a nearest neighbour in a k -d tree has the following procedure [38]:

1. Starting with the root node, the algorithm moves down the tree recursively.
2. Once the algorithm reaches a leaf node, it saves that node point as the “current best”
3. The algorithm unwinds the recursion of the tree, performing the following steps:
 1. If the current node is closer than the current best, then it becomes the current best.
 2. The algorithm checks whether there could be any points on the other side of the splitting plane that are closer to the search point than the current best. In concept, this is done by intersecting the splitting hyperplane with a hypersphere around the search point that has a radius equal to the current nearest distance. Since the hyperplanes are all axis-aligned this is implemented as a simple comparison to see whether the distance between the splitting coordinate of the search point and current node is lesser than the distance from the search point to the current best.
 1. If the hypersphere crosses the plane, there could be nearer points on the other side of the plane, so the algorithm must move down the other branch of the tree from the current node looking for closer points, following the same recursive process as the entire search.
 2. If the hypersphere doesn't intersect the splitting plane, then the algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.
4. When the algorithm finishes this process for the root node, then the search is complete.

This module is useful when you need to search for a large number of coordinates, so a web **API** is not practical. Also has the advantage of not needing an Internet connection and the search algorithm has the complexity of $O(\log n)$, which makes it slightly better than a linear search of all of the points and saves computation by using squared distances to avoid computing square roots.

The result of this module is the city, country code and country name of a given latitude/longitude coordinate and its name is `reverse_geocode` [39]. It can be inaccurate because it doesn't distinguish land coordinates from sea coordinates, as it tries to search for

the closest neighbour, regardless of its location. To complement this module, a map is drawn because of this inaccuracy and to give more context than only a country name.

Find Country Function

1. Initialize country list
2. **for** each point:
3. Country name equals the result of Reverse Geocode Search function (point)
4. Add country name to country list
5. **return** country list

4.2 World Visualization

Given that the previous module does not ignore any coordinate and always tries to find a physical location, adding a visual component will provide the user more context about the results. This being said, a module was used to draw a two-dimensional map with the exact location of all the coordinates given by the previous process.

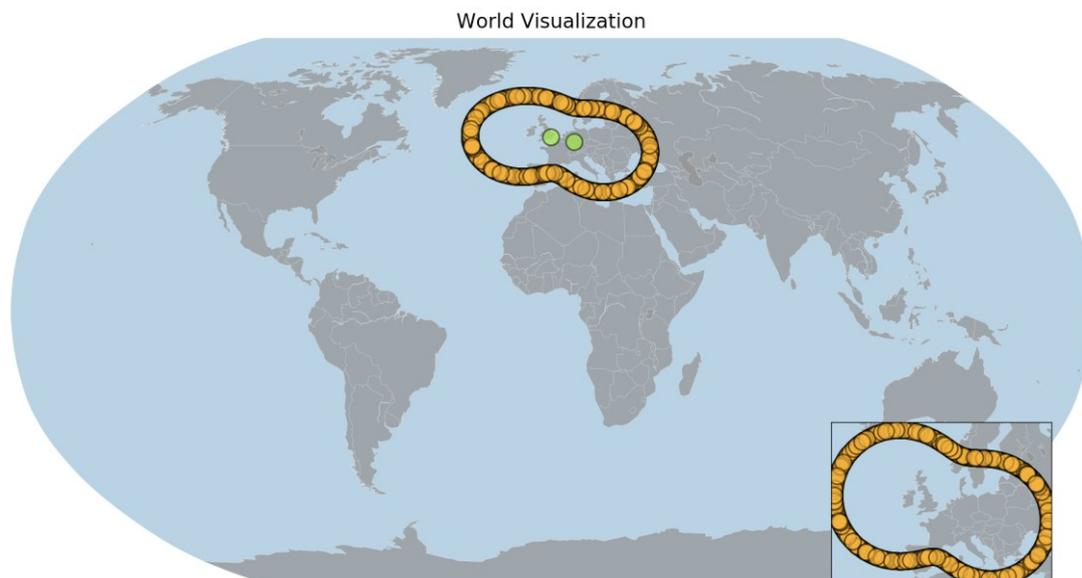


Figure 4.3 World Map with an Ellipse drawn

As shown in Figure 4.3, a two-dimensional map and mini-map, with a close-up view, are drawn. On the bigger map, the servers that have their traffic relayed are also drawn as green dots. The result coordinates are drawn in both map and mini-map as yellow dots, with the mini-map having a dynamic zooming depending on those coordinates. The colours to fill the background chosen were grey (#808080) for the continents and pale blue (#A6CAE0) for the oceans, since they don't annoy the eye and look neutral. These two colours allow the focus to be on the most important part, the results.

The reason a smaller map is drawn is that when the coordinates are close to each other they overlay themselves on the bigger map, and it's difficult to see which country they belong to.

The bigger map consists of a Robinson Projection ^[47]. This projection was specifically created in an attempt to find a good compromise to the problem of readily showing the

whole globe as a flat image. This projection doesn't preserve area measure nor angles. The meridians curve gently, avoiding extremes, but thereby stretch the poles into long lines instead of leaving them as points. Hence, distortion close to the poles is severe, but quickly declines to moderate levels away from them (Figure 4.4). This projection was previously used by the National Geographic Society.

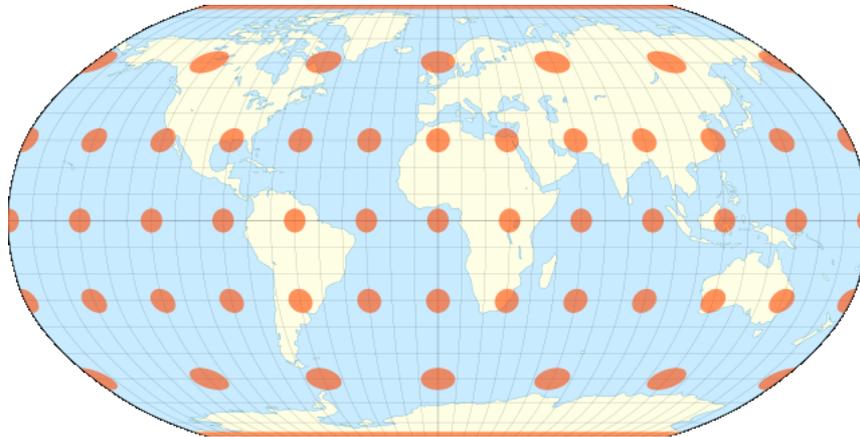


Figure 4.4 The Robinson Projection with Tissot's indicatrix of deformation ^[47]

Since the module does not allow for the Robinson Projection to be zoomed in, the smaller map consists of the Mercator Projection ^[46]. This projection is the standard map projection for nautical purposes because of the ability to represent lines of a constant course. The meridians are mapped to equally spaced vertical lines and circles of latitude are mapped to horizontal lines. Angles are preserved, and the size of the objects are distorted as the latitude increases from the equator to the poles. So, for example, landmasses such as Greenland and Antarctica appear much larger than they actually are relative to land masses near the equator, such as Central Africa ^[46] (Figure 4.5).

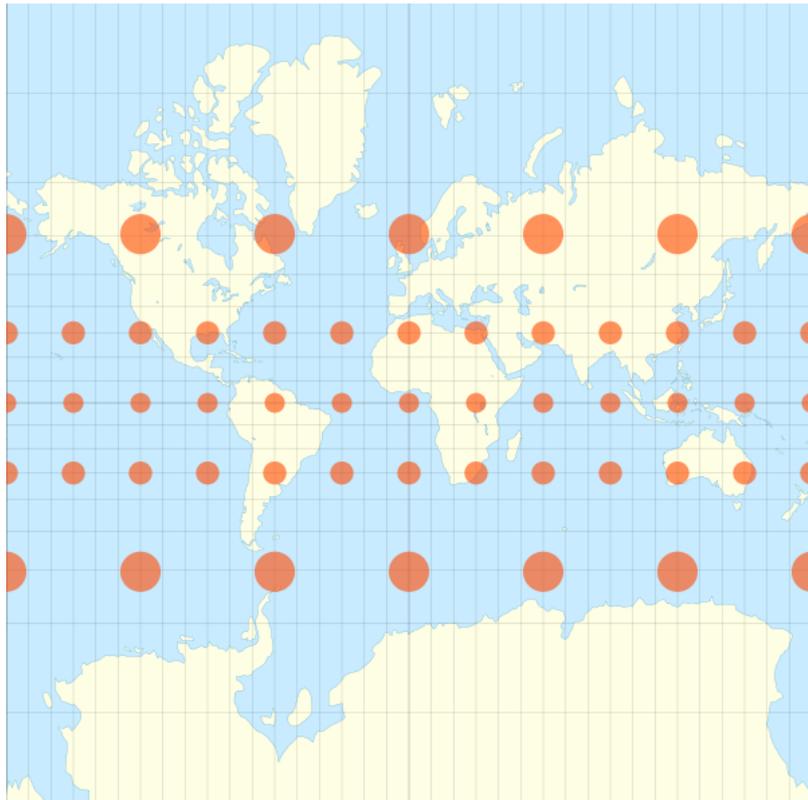


Figure 4.5 The Mercator Projection with Tissot's indicatrix of deformation ^[46]

This map is not shown entirely because it would defeat the purpose of a helping the user to see a close-up look. Instead, the boundaries are determined based on the points it will show, and it will zoom in accordingly. This is done by comparing coordinates and retrieving the minimum angle of latitude and longitude, and the maximum angle of latitude and longitude, of the array of coordinates to plot on the map. Each maximum value is added 1 degree and each minimum value is subtracted 1 degree as well, to give space on the map between the maximum or minimum latitudes/longitudes and the map boundaries.

The module used is Matplotlib Basemap ^[40]. It draws a map according to a given projection from their list. Their list doesn't have all the projections, although it has a fair amount of 24 different projections, each with its own advantage and disadvantages. This module is geared toward the needs of earth scientists and plots images, vectors, lines, points, shorelines, rivers and political boundaries. It was chosen because of the amount of details it contains.

Draw Map Function

1. Minimum Latitude is the calculation of minimum latitude of the points subtracted by one
2. Minimum Longitude is the calculation of minimum longitude of the points subtracted by one
3. Maximum Latitude is the calculation of maximum latitude of the points added by one
4. Maximum Longitude is the calculation of maximum longitude of the points added by one
5. Map equals Basemap function (Robin projection)
6. Draw Map boundaries
7. Draw Map continents
8. Draw Map coastlines
9. Draw Map countries
10. Plot Map
11. Mini Map equals Basemap function (Mercator projection with Minimum Latitude, Longitude and Maximum Latitude, Longitude)
12. Draw Mini Map boundaries
13. Draw Mini Map continents
14. Draw Mini Map coastlines
15. Draw Mini Map countries
16. Plot Mini Map
17. Show plot

4.3 Parsing Information and Data Errors

The user chooses which problem will be executed by using a command with the following parameters:

```
Python TeseF.py -c <path1> -p <path2> -n X
```

<path1> is the path to the non-relayed connections between servers.

<path2> is the path to the relayed connections between servers or a server and an unknown host. X is a number that can only take two values and chooses which problem will be executed. The values are:

- 1 for solving the Host Localization problem.
- 2 for solving the **BGP MITM** Attack Localization problem.

The algorithm requires a specific format to start working on both problems. Namely, two folders, but the most important part are the files that those folders contain. They must be in .tsv format and, depending on the folder, with the following names:

Folder	Host Localization	BGP MITM Attack Localization
Non-Relayed Information	<ServerName1>_<ServerName2>_NORELAY.tsv	Exactly the same as the Host Localization
Relayed Information	<ServerName>_<X>.tsv *X can be anything	<ServerName1>_<ServerName2>_<X>.tsv *X can be anything *ServerName1 can be equal to ServerName2

Table 4.1 File format

If a file breaks any of the following conditions, it will be ignored:

- a file does not obey into name and file format described in the table 4.1
- a server name must be either of the described in table 4.2

Following these two conditions will result in information about latency measurements from servers which their physical location is known.

After gathering all the files that provide non-relayed information about latency measurements, the information itself is collected. These files have multiple lines of data and must contain the attributes of a ping ^[37] (*Mini, Max, Avg, StdDev, Median* in this particular order) and respect the file format (information separated by tabs). This data is then grouped by an ascending order of their attribute Avg, using the Timsort ^[36] algorithm.

Timsort is a hybrid stable sorting algorithm that finds subsequences of the data that are already ordered and uses that knowledge to sort the remainder more efficiently. In each run it looks for two or more non-descending elements or descending elements. The descending elements are reversed and merged to other runs. It's the python standard algorithm for the sort function.

Next, the median of the data's **Min** attribute is calculated and saved as the data best entry. This processed is followed by calculating the data errors, based on the concept of the Probability Density Function ^[35]. The concept is that given a sample of values, there's a probability of a random variable falling within a particular range of values. In this case, the algorithm analyses each entry and will try to discover any errors, by matching their attributes values with the best entry. If any value oscillates more than 70% of the best entry value, it is considered an error. The error of a sample uses the following formula,

$$Error = \frac{MinErrorCount + MaxErrorCount}{TotalCount}$$

TotalCount is the total number of latency measurements, *MinErrorCount* is the number of data samples that have the attribute **Min** lower than 30% or higher than 170% of the best entry's **Min** and *MaxErrorCount* is the number of data samples that are within the 70% **Min** margin, but the **Max** attribute is lower than 30% of higher than 170% of the best entry's **Max**.

Tainted Results Function

1. Initialize error counter to zero
2. Initialize total counter to zero
3. **for** every ping:
4. Add one to the total counter
5. Minimum value is the ping minimum multiplied by 0.3
6. Maximum value is the ping maximum multiplied by 1.7
7. **if** the ping minimum is not between Minimum and Maximum value:
8. Add one to the error counter
9. **else if** the ping maximum is not between Minimum and Maximum value:
10. Add one to the error counter
11. **return** error counter divided by total counter

The best entry then is added to a list with its error and the sender (and receiver in the **BGP MITM** Attack Localization problem), and the connection between the two internet entities is registered. In case there's a file with information of an already registered connection, the previous process is exactly the same but instead of adding the best entry right away, there's a comparison between errors. The entry that has the smallest error is the one that replaces the other entry or stands in the list. This list will be used to calibrate the servers, as explained later on.

1. Initialize counter to 0
2. Initialize temporary array
3. **for** every line **in** a file:
4. Add ping data to the temporary array
5. Add one to the counter
6. Sort temporary array by ascending average
7. Error is the result of the Tainted Results Function
8. **return** a Ping **with** the median of all the temporary array attributes, the error, the receiver and the sender

The files that contain relayed information do not go through the same process. Their information must also contain the attributes of a ping ^[37] and they're added directly into a list that will be used by the algorithm.

4.4 Test Servers

The Test Servers are based on an internet graph^[34] made by Paulo Salvador as landmarks, being them the following:

City	Country	Latitude	Longitude	ServerName
Amsterdam	Netherlands	52.3702157	4.8951679	Amsterdam
Chicago	US	41.8781136	-87.6297982	Chicago
Frankfurt	Germany	50.1109221	8.6821267	Ger
Hafnarfjordur	Iceland	64.0291054	-21.9684626	Iceland
Hong-Kong	China	22.396428	114.109497	HongKong
Johannesburg	South Africa	-26.2041028	28.0473051	SouthAfrica
L.A.	US	34.0522342	-118.2436849	LA
London	UK	51.5073509	-0.1277583	London

Madrid	Spain	40.4167754	-3.7037902	Madrid
Milan	Italy	45.4654219	9.1859243	Milan
Moscow	Russia	55.755826	37.6173	Moscow
São Paulo	Brazil	-23.5505199	-46.6333094	SaoPaulo
Stockholm	Sweden	59.3293235	18.0685808	Sweden
Tel Aviv	Israel	32.0852999	34.7817676	Israel
Viña del Mar	Chile	-33.0153481	-71.5500276	Chile

Table 4.2 Landmarks

The language chosen to implement the algorithm is Python 2.7 because of the high compatibility with mathematical functions, google API's and plotting functions.

4.5 Results

Each problem was divided by two categories: Intercontinental, meaning it involves connection between servers in different continents, and Intracontinental, meaning it only involves connection between servers in the same continent. The files used to test are files generated by a Modular platform for detection of **BGP** routing attacks [32] where the solution to the problems is known, therefore there is a limit combinations of connections to use.

The results consist of a table with the solution, accuracy, connection and its ranking, the default rank being 1 (the best rank) and it's only mentioned if it's different than this value, results and a map with the results drawn. Accuracy is divided into Country Accuracy and Continent Accuracy. Each problem has its own way for calculating the accuracy.

4.5.1. Host Localization

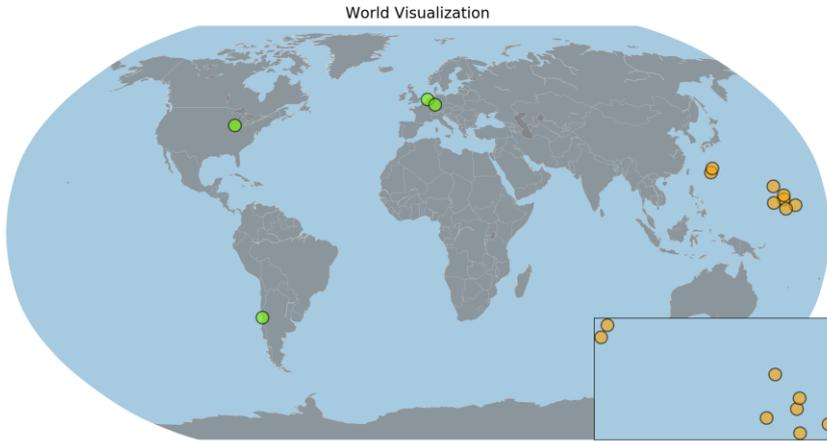
This problem takes a longer time to calculate the final results than the **BGP MITM** Attack Localization problem because it compares distances between every point of a circle and every point of the other drawn circles. Since it always chooses two points of every circle, some points have less accuracy and will seem off context, specially at Long Range results.

The accuracy of this problem is calculated using the following formula,

$$Accuracy(\%) = \frac{AccuratePoints}{TotalPoints}$$

AccuratePoints are points that belong to the same continent or country, depending on the accuracy pretended. The points with their physical location in the ocean but have their country name in the results are considered as inaccurate. The *TotalPoints* are easy to calculate because each connection used provides 2 points.

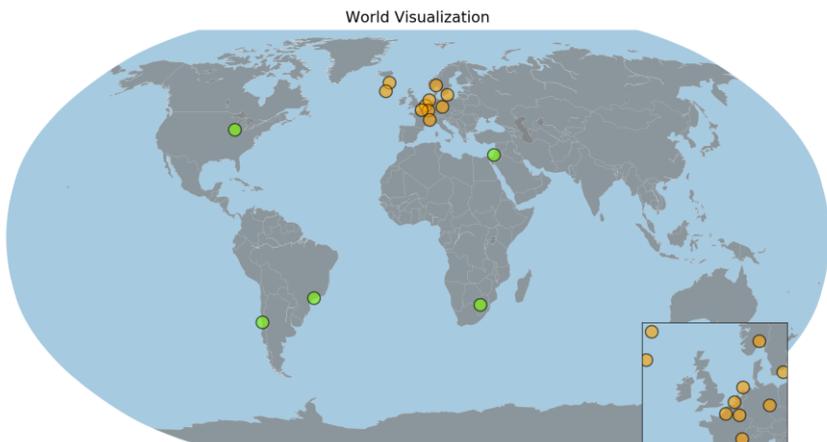
4.5.1.1. Intercontinental Connections



Solution:
Hong-Kong
Country
Accuracy 0%
Continent
Accuracy 0%
(Rank)
Connections:
Amsterdam - ?
Chicago - ?
(2) Chile - ?
Germany - ?

Results:

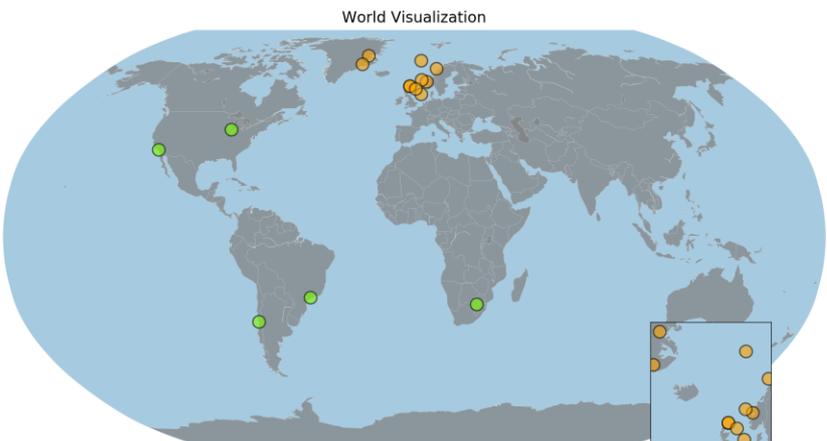
'Japan', 'Marshall Islands', 'Micronesia, Federated States of'



Solution:
London
Country
Accuracy 0%
Continent
Accuracy 80%
(Rank)
Connections:
(2)
SouthAfrica-?
Israel-?
(2) Chile-?
Chicago-?
(2) SaoPaulo-?
?

Results:

'Netherlands', 'Iceland', 'France', 'Sweden', 'Germany', 'Norway'



Solution:
Germany
Country
Accuracy 0%
Continent
Accuracy 40%
(Rank)
Connections:

(2) SouthAfrica-?

LA-?

(2) Chile-?

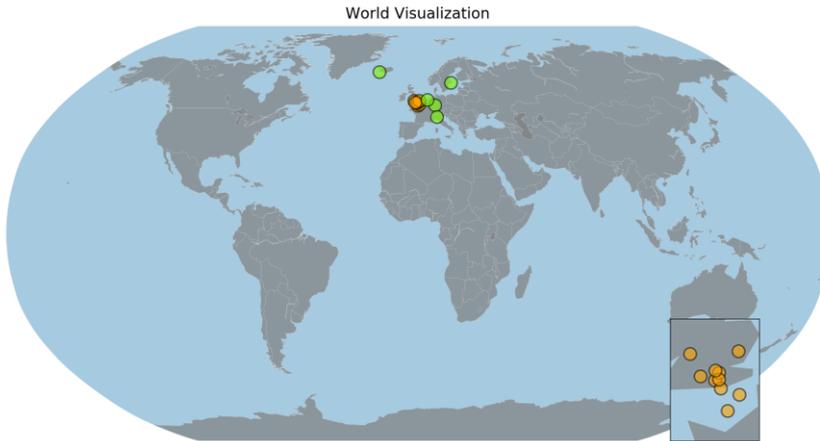
Chicago-?

(2) SaoPaulo-?

Results:

'United Kingdom', 'Iceland', 'Netherlands', 'Norway', 'Greenland'

4.5.1.2. Intracontinental connections



Solution:

London

Country Accuracy
80%

Continent Accuracy
80%

(Rank) Connections:

Amsterdam-?

Germany-?

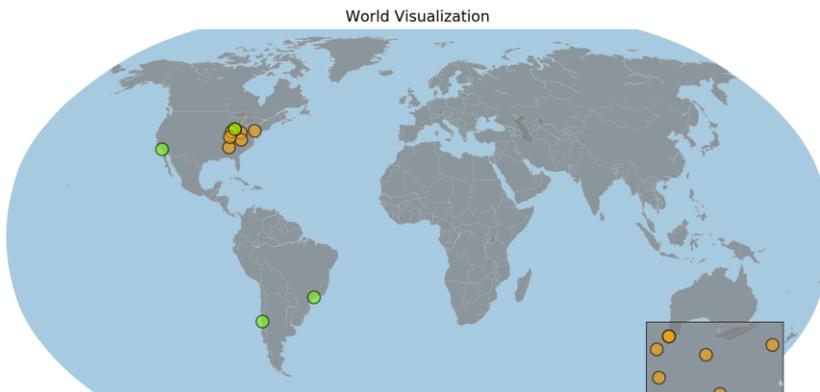
Milan-?

Sweden-?

Iceland-?

Results:

'United Kingdom', 'France'



Solution:

Chicago

Country Accuracy
100%

Continent Accuracy
100%

(Rank) Connections:

Chicago-?

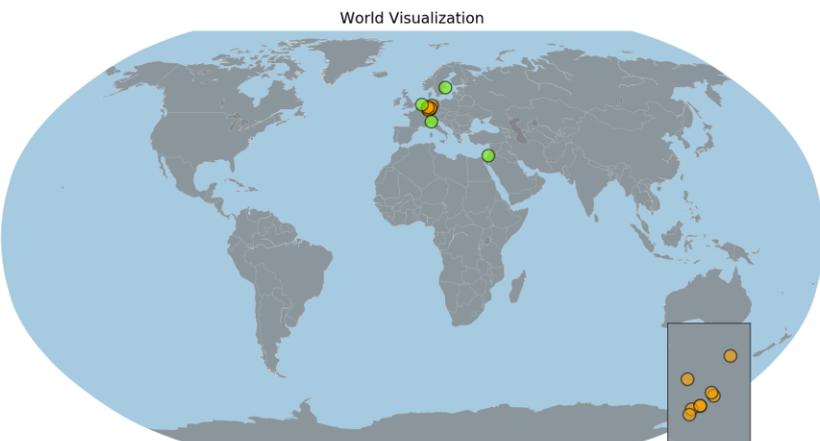
(2) SaoPaulo-?

LA-?

(2) Chile-?

Results:

'United States'



Solution:

Germany

Country Accuracy 100% Continent Accuracy 100%

(Rank) Connections:

Israel-?

Sweden-?

Milan-?

Amsterdam-?

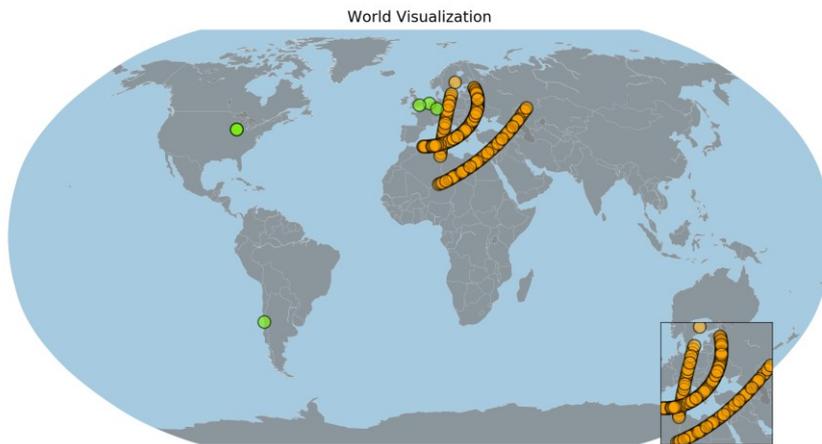
Results:

'Germany'

4.5.2. BGP MITM Attack Localization

There is no limit of drawn points in this problem unlike the previous problem. This can be very accurate at small range if the continent has a lot of servers. At long range the accuracy is affected specially if there are rank 2, or more, connections. Since there are no ways to calculate the *TotalPoints*, due to not being able to see if a point is in the ocean or not, the accuracy is calculated by using the result country list. It leads to an inaccurate method of accuracy and it's strongly advised to look at the context and not just the accuracy to understand the results.

4.5.2.1. Intercontinental connections



Solution:

Moscow

Country Accuracy 4%
Continent Accuracy 68%

(Rank) Connections:

Amsterdam-London

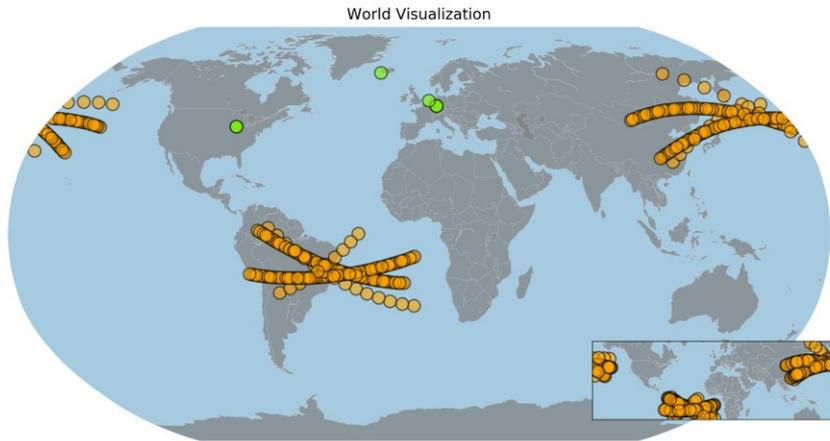
Chicago-Germany

(2) Chile-Chicago

Results:

'Turkey', 'Libyan Arab Jamahiriya', 'Italy', 'Czech Republic', 'Aland Islands', 'Belarus', 'Algeria', 'Germany', 'Russian Federation', 'Kazakhstan', 'Moldova, Republic of', 'Ukraine', 'Georgia', 'Poland', 'Macedonia', 'Sweden', 'Bulgaria', 'Romania', 'Albania', 'Chad', 'Tunisia', 'Egypt', 'Austria', 'Niger', 'Cyprus'

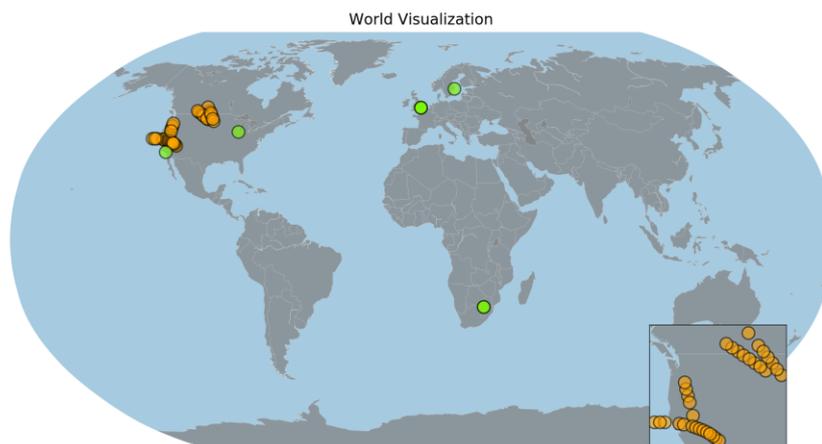
Chapter 4 – Geolocalization Results



Solution:
SaoPaulo
Country Accuracy
7,14%
Continent Accuracy 50%
(Rank)
Connections:
Iceland - Ger
Chicago - Chicago
Ger - Ger
Amsterdam - Chicago

Results:

'Brazil', 'Colombia', 'Saint Helena', 'Korea, Republic of', 'Mongolia', 'United States', 'Paraguay', 'China', 'Peru', 'Russian Federation', 'Japan', 'Argentina', 'Bolivia', 'Venezuela'

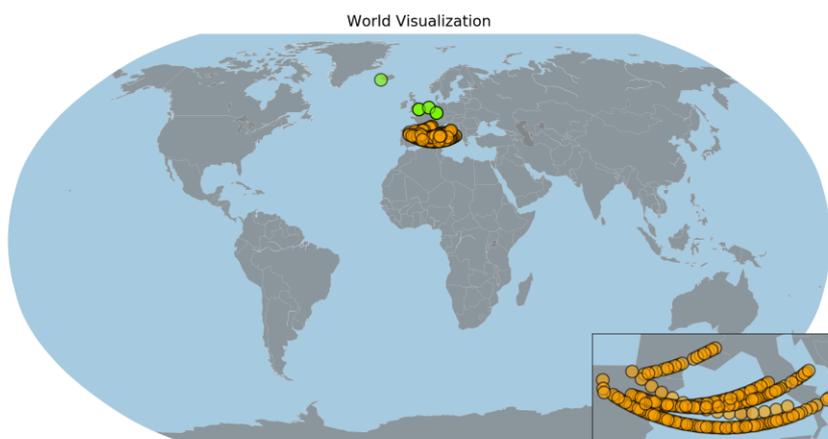


Solution:
LA
Country Accuracy 50%
Continent Accuracy 100%
(Rank)
Connections:
(2) SouthAfrica - Chicago
(2) SouthAfrica - London
Sweden - London
LA - London

Results:

'Canada', 'United States'

4.5.2.1. Intracontinental connections



Solution:
Madrid
Country Accuracy 20%
Continent Accuracy 80%
(Rank)
Connections:
Amsterdam - Germany

Chapter 4 – Geolocalization Results

Amsterdam-London

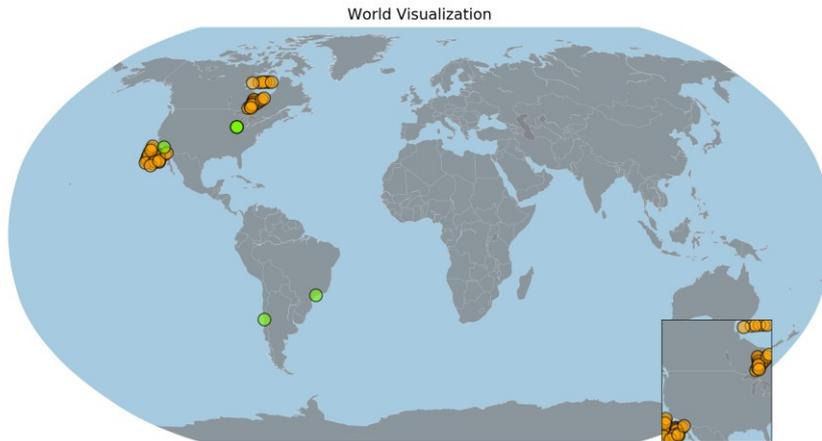
Germany-Germany

Germany-London

Iceland-London

Results:

'Croatia', 'Italy', 'Algeria', 'Spain', 'France'



Solution:

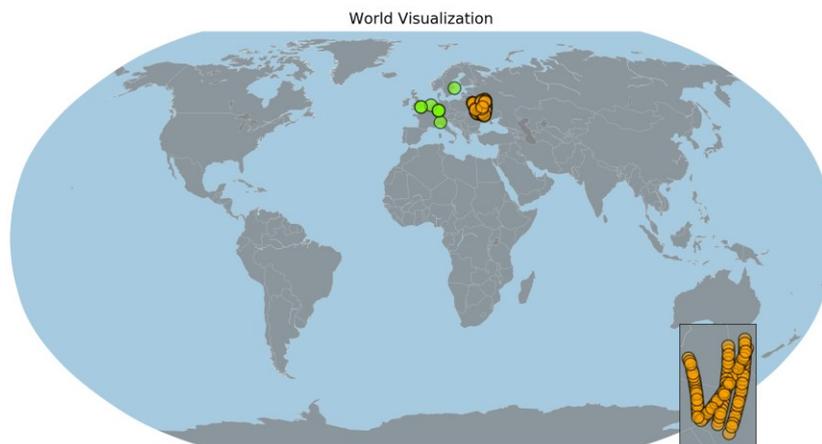
LA

Country Accuracy
33,33%
Continent Accuracy
100%

(Rank) Connections:
(2) SaoPaulo - Chicago
LA-Chicago
(2) Chile - Chicago

Results:

'United States', 'Canada', 'Mexico'



Solution:

Moscow

Country Accuracy
33,33%
Continent Accuracy
100%

(Rank) Connections:
Sweden - London
Milan - Ger
Amsterdam - Ger
Ger - London

Results:

'Ukraine', 'Russian Federation', 'Belarus'

4.6 Result Discussion

The solution to the Host Localization problem at long ranges is highly inaccurate Country-wise but has a decent Continent Accuracy. At short ranges is highly accurate Continent-wise and a has a high-Country Accuracy.

Intercontinental	
Hong-Kong	0%
	0%
London	0%
	80%
Germany	0%
	40%
Intracontinental	
Germany	100%
	100%
London	80%
	80%
Chicago	100%
	100%

Table 4.3 Host Localization problem results

The solution to the **BGP MITM** Attack Localization is fairly accurate at long ranges and has a decent Country Accuracy at short ranges.

Intercontinental	
LA	50%
	100%
SaoPaulo	7,14%
	50%
Moscow	4%
	68%
Intracontinental	
Moscow	33%
	100%
LA	33%
	100%

Madrid	20%	
	80%	

Table 4.4 BGP MITM Localization problem results

All of these accuracies are highly dependent on the connections used. The **BGP MITM** Attack Localization solution looks more accurate at long ranges and less accurate at short ranges because of the used method of calculating the accuracy. This results with context are more accurate.

Both solutions are more successful with intracontinental connections, as expected, due to the distance between the servers being smaller and with less errors. The solution to the Host Localization problem is more accurate at short distances, while the solution to the **BGP MITM** Attack Localization problem is more accurate at long distances.

This being said, the results are satisfactory at short distances and acceptable at some situations at long distances, less acceptable with the Host Localization solution.

5 Conclusion

This dissertation proposed a solution to finding an unknown host and a solution to locating an internet host that is redirecting **BGP** traffic from two servers communicating with each other. They both need **.tsv** files to feed the algorithm. The more connections and servers to work with, the better and accurate the results are due to having more information and more area covered by the servers. Without any information the algorithm would not be able to find any host. With that in mind, the algorithm gathers information before anything is done.

The goal of this dissertation was accomplished but some of the long-range results are not accurate. It also heavily depends on the information provided and nothing is static, so the user has some of the responsibility for the accuracy.

Although the goal was met, there is still room for improvement however. The error calculation does not take terrain, depth and elevation into account. Distance between the servers are not as direct as the algorithm calculates. The algorithm assumes a link between each city is a direct link which is not the case (Figure 5.1). The database doesn't exist since it is created in the moment, with the available data, and nothing is saved.

This being said, improvements could be:

- Making a Graphic User Interface
- Account for more variables in the error calculation, such as terrain elevation and depth
- Distance between cities also be accounted by cable connections of **ISP's**
- New type of formats on gathering information
- Creating a database to save the already known information

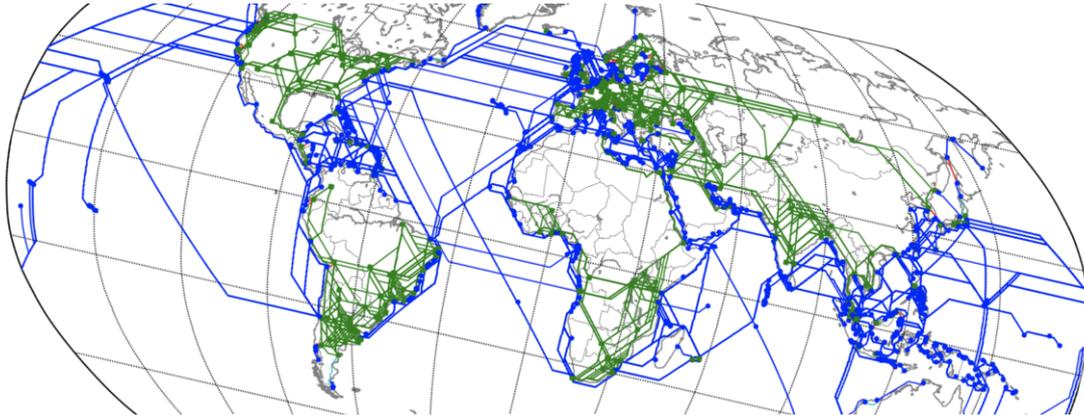


Figure 5.1 A Quasi-Realistic Internet Graph [34]

References

- [1] "Geolocation", May 2018. <<https://en.wikipedia.org/wiki/Geolocation>>
- [2] V. N. Padmanabhan, L. Subramanian. "Determining the Geographic Location of Internet Hosts", *Microsoft Research Technical Report MSR-TR-2000-110*, November 2000.
- [3] Ashtarifar, S., Matrawy. A. "Determining Host Location on the Internet: The Case of VoIP Emergency Calls", *Communications. Workshops, 2009. ICC Workshops 2009. IEEE International Conference*, pp. 1-5.
- [4] H. Li, Y. He, R. Xi. "A complete evaluation of the Chinese geolocation databases", *Proceedings of 2015 8th International Conference on Intelligent Computation Technology and Automation*, pp. 13-17, 2015.
- [5] Peter Hillmann, Lars Stiemert, Gabi Dreo Rodosek, Oliver Rose. "Dragoon: Advanced Modelling of IP Geolocation by use of Latency Measurements", *In Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST-2015)*. IEEE, 2015.
- [6] "IPLocation", May 2018. <<https://www.iplocation.net/>>
- [7] "WhoIs", May 2018. <<https://www.whois.net/>>
- [8] "IPInfo", May 2018. <<http://www.ipinfo.io/>>
- [9] James A. Muir, Paul C. Van Oorschot. "Internet geolocation: Evasion and counterevasion", *ACM Computing Surveys (CSUR)*, v.42 n.1, p.1-23, December 2009
- [10] Internet Engineering Taskforce Geographic Location/Privacy Working Group, May 2018. <<https://www.datatracker.ietf.org/wg/geopriv/documents/>>
- [11] "Maxmind", May 2018. <<https://www.maxmind.com>>
- [12] "Db-IP", May 2018. <<https://www.db-ip.com>>
- [13] "Planet-Lab", May 2018. <<https://www.planet-lab.org>>
- [14] "IP2Location", May 2018. <<http://www.ip2location.com>>
- [15] "LOC record", May 2018. <https://en.wikipedia.org/wiki/LOC_record>
- [16] "Regional Internet Registry", June 2018. <https://en.wikipedia.org/wiki/Regional_Internet_registry>
- [17] "WhatRoute", May 2018. <<https://www.whatroute.net/index.html>>
- [18] "VisualRoute", May 2018. <<http://www.visualroute.com>>
- [19] "Akamai", May 018. <<https://www.akamai.com>>
- [20] "DigitalEnvoy", May 2018. <<https://www.digitalenvoy.com>>
- [21] "Neustar", May 2018. <<https://www.neustar.biz>>
- [22] "F5", May 2018. <<https://f5.com>>
- [23] "Verifia", May 2018. <<https://www.securitywatchdog.org.uk/due-diligence>>
- [24] "GeoPoint", May 2018. <<http://www.geopoint.pt/pt/>>

- [25] Ethan Katz-Bassett, John P. John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, Yatin Chawathe. "Towards IP geolocation using delay and topology measurements", *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, October 25-27, 2006, Rio de Janeiro, Brazil
- [26] Bamba Gueye, Artur Ziviani, Mark Crovella, Serge Fdida. "Constraint-based geolocation of internet hosts", *IEEE/ACM Transactions on Networking (TON)*, v.14 n.6, p.1219-1232, December 2006
- [27] B. Wong, I. Stoyanov, and E. G. Sirer. "Octant: A comprehensive framework for the geolocalization of Internet hosts", in *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'07. USENIX Association, 2007, pp. 23-23.
- [28] B. Eriksson, P. Barford, B. Maggs, and R. Nowak. "Posit: An adaptive framework for lightweight ip geolocation", BU/CS, Tech. Rep., July 2011.
- [29] A. Dahnert. "Hawkeyes: An advanced ip geolocation approach: Ip geolocation using semantic and measurement-based techniques", in *Cybersecurity Summit (WCS), 2011 Second Worldwide*, June 2011.
- [30] S. Laki, P. Matray, P. Haga, T. Sebok, I. Csabai, and G. Vattay, "Spotter: A model based active geolocation service", in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 3173-3181.
- [31] Paulo Salvador and António Nogueira. "Customer-Side Detection of Internet-Scale Traffic Redirection", *Telecommunications Network Strategy and Planning Symposium (Networks)*, 2014 16th International
- [32] Marco Filipe Moutinho da Silva, "Modular platform for detection of BGP routing attacks", IT, Aveiro
- [33] "CAIDA", May 2018.
<<http://www.caida.org/projects/cybersecurity/geolocation/>>
- [34] Paulo Salvador, "A Quasi-Realistic Internet Graph", *8th International Conference on data Communication Networking (DCNET 2017)*, 24-26 July, Madrid, Spain.
- [35] "Probability Density Function", May 2018.
<https://en.wikipedia.org/wiki/Probability_density_function>
- [36] "Timsort", May 2018. <<https://en.wikipedia.org/wiki/Timsort>>
- [37] "Ping", May 2018. <[https://en.wikipedia.org/wiki/Ping_\(networking_utility\)](https://en.wikipedia.org/wiki/Ping_(networking_utility))>
- [38] "K-d tree", May 2018. <https://en.wikipedia.org/wiki/K-d_tree>
- [39] "Reverse Geocode", May 2018. Python Library.
<https://pypi.org/project/reverse_geocode>
- [40] "Basemap", May 2018. Python Matplotlib. <<https://matplotlib.org/basemap>>
- [41] "Azimuth", May 2018. <<https://en.wikipedia.org/wiki/Azimuth>>
- [42] "Ellipse", May 2018. <<https://pt.wikipedia.org/wiki/Elipse>>
- [43] "Pythagorean Theorem", May 2018.
<https://en.wikipedia.org/wiki/Pythagorean_theorem>
- [44] "World Geodetic System", June 2018.
<https://en.wikipedia.org/wiki/World_Geodetic_System>
- [45] "Geographiclib", June 2018.
<<https://geographiclib.sourceforge.io/html/python/>>
- [46] "Mercator Projection", June 2018.
<https://en.wikipedia.org/wiki/Mercator_projection>
- [47] "Robinson Projection", June 2018
<https://en.wikipedia.org/wiki/Robinson_projection>
- [48] "Reverse Geocoding", June 2018
<https://en.wikipedia.org/wiki/Reverse_geocoding>
- [49] "EurekaAPI", June 2018 <<https://www.eurekapi.com/>>
- [50] "IPLigence", June 2018 <<http://www.ipligence.com/>>
- [51] "IpInfoDB", June 2018 <<https://ipinfodb.com/>>

[52] “Geolocation Providers”, June 2018
<<https://whatismyipaddress.com/geolocation-providers>>