UCLouvain

epl

École polytechnique de Louvain

Generating maps of European Internet Service Providers

Author: **Elise PEIFFER**
Supervisor: **Cristel PELSSER**
Readers: **Ramin SADRE, François MICHEL**
Academic year 2023–2024
Master [60] in Computer Science

Contents

Acronyms

AS Autonomous System. 3, 4, 5, 6, 7, 8, 19, 32, 46,

ASN Autonomous System Number. 6, 12, 13, 23, 24, 25, 32, 33, 39, 40, 45, 46

BGP Border Gateway Protocol. 6, 8

IP Internet Protocol. i, 4, 5, 6, 7, 8

ISP Internet Service Provider. 3, 6, 13, 23, 32, 34, 35

IX Internet Exchange. 6, 9, 10, 13, 15, 16, 19, 23, 24, 25, 26, 39, 40, 45, 46

MAN Metropolitan Area Network. 16, 17

PoP Point of Presence. 12, 16, 17, 18, 36

TCP Transmission Control Protocol. 4

UDP User Datagram Protocol. 4

Acknowledgments

I would like to start by expressing my deepest gratitude to my supervisor, Pr. Pelsser, for her patience and guidance throughout the development of this project. Her insightful feedback and continued support helped shape this work.

I am also grateful to my family for their encouragement, especially my father and brother for their precious feedback and unwavering support.

Lastly, I extend special thanks to my friends who supported me through stressful times. I would also like to mention my cats, who provided many hours of entertainment and companionship.

Introduction

The primary objective of this thesis is to consolidate data concerning Internet backbones networks that span the European continent. This data encompasses various facets, including network ownership, technology, capacity, routes, and interconnectivity between networks. Having a comprehensive repository of information on European network infrastructure is vital for a multitude of research endeavors and practical applications. There currently exists no single, consolidated, and comprehensive source for this information in Europe. This contrasts with the situation in the United States, where specialised data brokers offer such consolidated data.

Network backbone maps are valuable for a broad spectrum of investigative purposes across numerous fields.

For instance, when analysing telecommunications infrastructure, these maps enable detailed examination of the distribution, density, and connectivity of networks across Europe. This is crucial for understanding broadband coverage, identifying underserved areas, and assessing the overall development of telecommunications infrastructure.

In terms of security and resilience, network backbone maps are instrumental in evaluating the robustness of telecommunications infrastructure. Analysts can scrutinise redundancy, pinpoint potential single points of failure, and gauge the impact of network disruptions on critical services and industries.

For network performance investigations, these maps facilitate the exploration of metrics such as bandwidth and reliability. By studying the geographical distribution of routes and network interconnections, experts can determine how these factors influence performance and user experience.

These maps are also invaluable for examining the digital divide and disparities in broadband access across Europe. By integrating network infrastructure data with demographic and socioeconomic information, it becomes feasible to identify regions with limited high-speed internet access and develop strategies to bridge this digital divide.

Urban planners and policymakers can utilise network maps to inform infrastructure development and urban planning initiatives. This enables the optimisation of new projects, such as smart city initiatives or transportation networks, by strategically placing infrastructure.

In market analysis and competition studies, network backbone maps provide crucial insights into the competitive landscape of the telecommunications sector in Europe. Analysts can assess the deployment strategies of various network operators, evaluate market concentration, and identify areas of competition or collaboration.

The problem addressed by this thesis is the lack of comprehensive, consolidated

data. Information is dispersed and varies in detail, typically available only from individual network operators. For the European continent, data is less consolidated compared to the United States, where data brokers specialise in providing such information. This work focuses specifically on European continental network backbones, excluding metropolitan area networks and last-mile connectivity, which are more readily accessible from commercial data brokers.

In the course of this project, we have identified sources of information on European networks, developed a data model to consolidate the data, defined and tested a procedure for data collection, and visualised the repository data on maps. This thesis will discuss the methods and tools that we conceived to realise the desired data consolidation, the challenges encountered, and propose solutions for the short and medium term.

Chapter 1

Backbone Networks

Before delving into the detail of how we have approached the problem of consolidating data regarding backbone networks covering the European continent, let's first understand why these backbones exist and how they are operated.

1.1 The Internet

An easy way to describe the Internet is by defining it as a network of interconnected computers and other devices that spans the globe. It is a decentralised network, meaning that it has no central authority controlling it. Instead, it is made up of thousands[1] of smaller networks, called Autonomous Systems (ASes), operated by various organisations, including governments, businesses, educational institutions, and individuals.

At its core, the Internet consists of high-speed data connections which interconnect major regions and cities around the world. These backbone networks are operated by telecommunications companies, Internet Service Providers (ISPs), and other organisations. They use a combination of fiber optic cables, satellite links, and other technologies to transmit data over long distances. Intercommunication between two separate providers is achieved by both organisations connecting in Internet exchange facilities [1].

Connecting to the Internet requires a device with networking capabilities, such as a computer, smartphone, tablet, or internet-enabled appliance. During a communication session between two or more interconnected devices, the transport of data between the endpoint devices partaking in the communication (source host and destination device(s)) is organised by a set of standardised protocols like the

[1] As of May 4th 2024, there are 64.493 16-bit allocated ASes, and 59.408 32-bit allocated ASes (https://www.potaroo.net/tools/asn32/index.html).

Transmission Control Protocol (TCP) or User Datagram Protocol (UDP)[2].

Data transmitted over the Internet is broken down into smaller units called packets, which are routed across the network from the source device to the destination device, using the Internet Protocol (IP). Routers and switches are responsible for directing these packets along the most efficient paths through the network, based on factors such as network congestion and the availability of alternative routes.

The study of the available paths betwen any two points, any source and any destination, is one of the reasons why it is useful to have comprehensive data on the backbone networks, the subject of this thesis.

1.2 Autonomous Systems

In the previous section, we have described the Internet as a gigantic globe-spanning network of interconnected computers and internet-communicating devices. The topology of that network can be described as a complex mesh of interconnected smaller networks, whereby the connections between smaller networks can be referred to as backbones and the larger network comprising the backbone connections as backbone network. That same pattern – a mesh network of smaller networks, with backbone connections connecting the networks – is repeated at several geographical levels. At the lowest level, the last-mile distribution, other network topologies like star and tree networks exist (see Figure 1.1 for the basic layouts of the various network topologies).

The networks that comprise the Internet are called Autonomous Systems (ASes). Every computer or device that connects to the Internet does so through an AS. An AS is a subnetwork of the Internet that groups all IP addresses with a common initial part, known as a common IP routing prefix[3]. More formally and generally [3], an AS is an IP routing prefix (or a collection of IP routing prefixes) managed by one or more network operators that presents a unified, clearly defined routing

[2]These protocols handle data transport over the network from the source to the final destination, regardless of the data volume. The first protocol is more reliable, ensuring correct data sequencing at the reception side, incorporating mechanisms to recover from data loss, and controlling the data flow to prevent overwhelming the recipient. In contrast, the second protocol is more efficient, as it initiates connections faster, delivers data with lower latency, and uses fewer resources by avoiding back-and-forth synchronisation and acknowledgments between sender and receiver. TCP is ideal for point-to-point transmissions requiring receipt confirmation in applications where data integrity matters more than transmission speed. On the other hand, UDP is suitable for multicast and broadcast to multiple receivers in real-time applications where rapid transmission is essential, and occasional data packet loss is acceptable.

[3]In IP-based computer networks, a routing prefix can be seen as the unvarying portion of the IP address shared by each client on a single subnet.

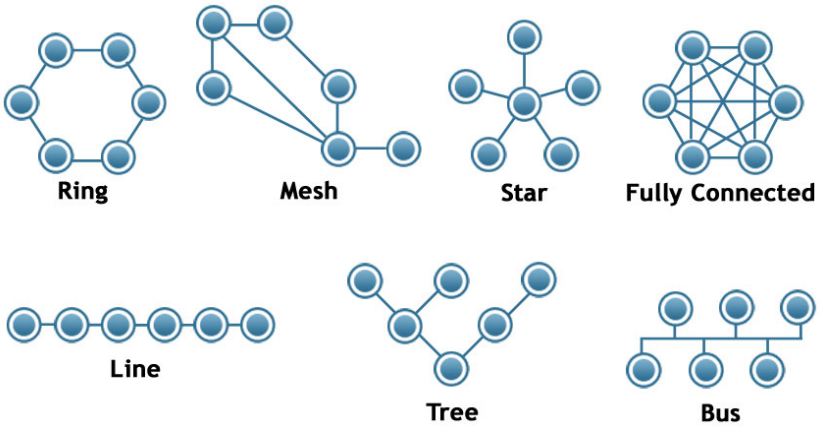


Figure 1.1: Types of network topologies (image from [2]).

policy to the Internet[4]. This means that all packets originating from the rest of the Internet and destined for an IP address that matches with an AS's routing prefix will be directed to that AS, which will then ensure the packets reach the correct individual recipients within its network.

Figure 1.2: Network of networks (image from [3]).

[4]This routing policy is a list of the IP address space that the AS controls, plus a list of the other ASes to which it connects. This information is necessary for routing packets to the correct networks.

An AS typically represents a single organisation or entity[5], such as an Internet Service Provider (ISP), a large enterprise, or a university network.

Each Autonomous System is assigned a unique identifier called an Autonomous System Number (ASN), which is used to identify it on the internet. ASNs are globally unique and assigned by regional internet registries (RIRs) such as ARIN (American Registry for Internet Numbers), RIPE NCC (Réseaux IP Européens Network Coordination Centre), and APNIC (Asia-Pacific Network Information Centre). All ASes together constitute the Internet; making it a network of networks (symbolically depicted in Figure 1.2).

Autonomous Systems are a fundamental building block of the internet's routing infrastructure. They use exterior gateway protocols, such as the Border Gateway Protocol (BGP), to exchange routing information with other Autonomous Systems. BGP allows ASes to advertise the IP prefixes they can reach and to learn about reachable IP prefixes from other ASes. It is in this sense that BGP is referred to as the postal service of the Internet.

ASes play a crucial role in internet routing, as they define the boundaries within which routing decisions are made. They enable traffic to be directed efficiently across the internet by allowing networks to exchange routing information and determine the best path for data packets to reach their destination in a series of hops from AS to AS.

1.3 Peering

1.3.1 Peering

Peering refers to the arrangement between two Autonomous Systems to exchange traffic directly, rather than through a third-party network. This direct exchange of traffic allows ASes to efficiently route data between their networks, improving performance and reducing costs associated with using intermediary networks.

There are two main types of peering: public and private peering [4].

Public peering occurs at an Internet Exchange (IX), where multiple ASes connect to a shared infrastructure to exchange traffic. This setup allows for the efficient and cost-effective exchange of data among the many Autonomous Systems present in the IX.

Private peering involves a direct, point-to-point connection between two ASes, typically within a data center. Private peering is used when the volume of traf-

[5]The inverse isn't true; a single organisation can operate multiple ASes.

fic exchanged between the two networks is large enough to justify a dedicated connection.

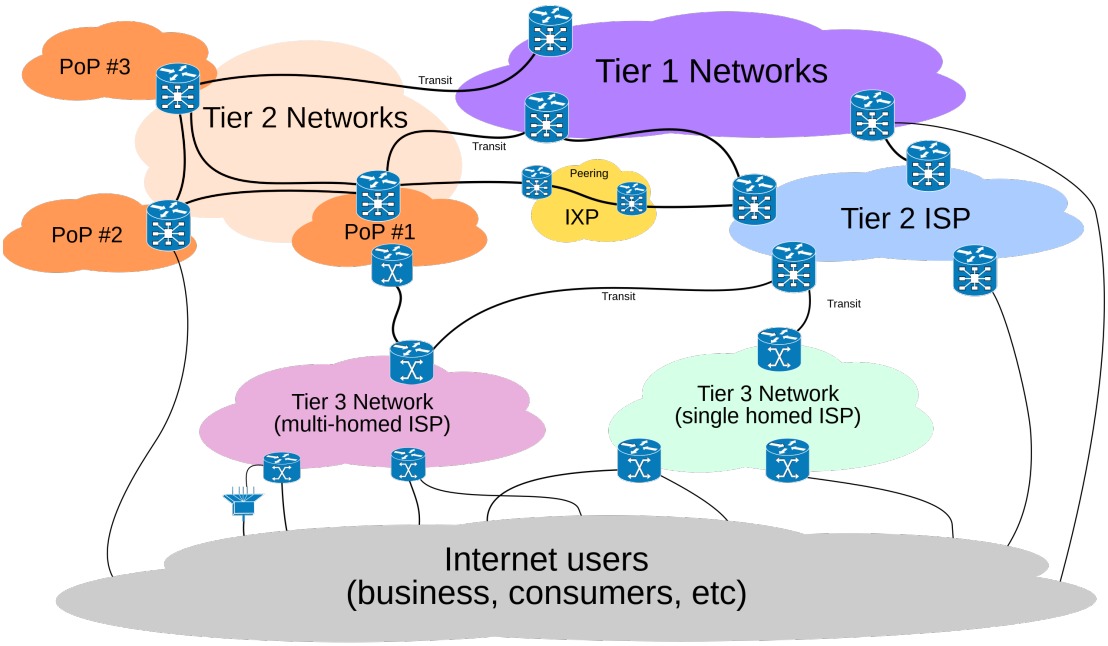


Figure 1.3: Relationship between Tiers of Internet providers (image from [5]).

1.3.2 Settlement-free Peering and IP Transit

When the traffic flows between two peering networks are (more or less) in balance, it is less expensive to allow both parties to peer on eachother's networks free-of-charge than to meter and charge. Both networks benefit by reaching more parties and no money needs to be spent on setting up metering and charging and doing all the administration that that entails. In such case it is said that both network operators do settlement-free peering.

In the absence of settlement-free peering agreements, if a network operator wishes to use the network of another network operator to reach another part of the Internet, it is said that the first operator purchases IP-transit from the second.

1.3.3 Network Tiering

A Tier-1 network is an Autonomous System or a group of collectively-managed ASes that can connect to every other network on the Internet exclusively through settlement-free peering [5]. These networks can exchange traffic with other Tier-1

networks without incurring any fees for the traffic exchange in either direction (this is often referred to as a peer-to-peer relationship). In contrast, a Tier-2 network engages in peering with other networks but also buys IP transit to access certain parts of the Internet. The term Tier-3 is occasionally used to describe networks that only purchase IP transit from other networks to access the Internet (these are referred to as customer/provider relationships, where Tier-3 is a customer in all its relationships, and Tier-2 can be customer or provider).

1.4 BGP Routing

As data travels across the internet from source to destination, every AS in between has to decide where the data packet should go next. For this purpose routers are used. Each of these devices has a list – known as the routing table – of possible pathways it can send the data packets over. When a packet arrives at a router node, a certain number of checks are performed to decide the best path for the packet. This decision is based on several factors like network congestion and data transfer cost. BGP routing[6] considers these factors and helps determine the next best Autonomous System so that data travels on the fastest and cheapest route from source to destination [6, 7].

Some of BGP's functions include [8]:

- Ensuring up-to-date information: by regularly updating a routing table with all the latest information on Internet routes, the routers can compute the best paths for the data packets they're transporting;
- Choosing the best paths and offering alternative paths: factors such as cost and time help determine which of the different available paths performs the best, and since there is a selection of available routes, this means that in case of route failure, BGP can offer an alternative path;
- Discovering loops: the protocol can detect when a route loops through a selection of algorithms (known as the BGP Decision Process), and can avoid these non efficient detours;
- Securing traffic: by verifying the origin of a BGP message is legitimate, and authenticating it with a predefined password or key, the protocol ensures the security of the data being transferred over the network;
- Communicating with other networks: the protocol ensures communication between all devices no matter the type of network they're a part of (an IPv4 device can thus communicate with an IPv6 router without any problem).

[6]When BGP runs between two nodes in the same Autonomous System (AS), it is referred to as Internal BGP (iBGP or Interior Border Gateway Protocol). When it runs between different Autonomous Systems, it is called External BGP (eBGP or Exterior Border Gateway Protocol).

Chapter 2

Sources of Information

In order to compile a comprehensive set of data concerning European backbone networks, various sources provide bits of information used in the context of this project. It is the combination of these different sources that allowed us to produce the results presented in this report.

These sources are: PeeringDB[1]; CAIDA[2]; and the official websites of Internet operators.

2.1 PeeringDB

PeeringDB is a "nonprofit member-based organization that facilitates the exchange of user maintained interconnection related information, primarily for Peering Coordinators and Internet Exchange, Facility, and Network Operators" [9].

It is an open-source service helping Internet operators define their peering strategy by providing interconnection information.

The primary purpose of PeeringDB is to serve as a central repository of data related to network peering. It allows network operators to publish details about their network infrastructure, such as the locations of their peering points (Internet Exchangees or private interconnects), the networks they peer with, and any specific peering policies they have in place.

PeeringDB offers an online user interface (example of the page of an IX in Figure 2.1) that allows users to do a look-up of any specific data element stored in the datase. For more structured queries Peering DB exposes an API.

Observing the API's structure helps with understanding the database's data model. Its main components are the *organisations* and their child entities: *facilities*, *networks*, and *internet exchanges* (IXes). Other entities, such as *ixfac*s and *netixlan*s

[1] https://www.peeringdb.com/
[2] https://www.caida.org/

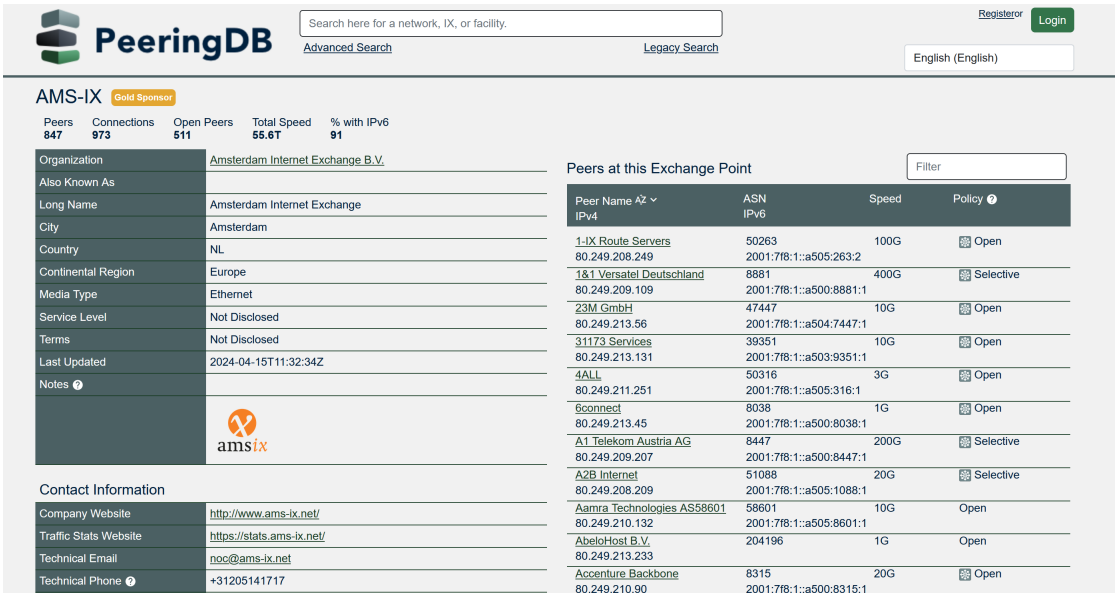


Figure 2.1: PeeringDB's user interface.

represent combinations of two of these child entities. An *ixfac* type object is derived from the intersection between a *fac* object and an *ix* object; in other words: an IX's presence at a certain interconnection facility is represented by an *ixfac* type object in the database [10]. The same reasoning applies to *netfac* and *netixlan* objects.

The schema below is a simplified visualisation of the elements pertinent in the context of this project from PeeringDB's structure[3].

[3]For the sake of the conciseness, we only added primary and foreign keys as attributes; other attributes will be adressed later on.

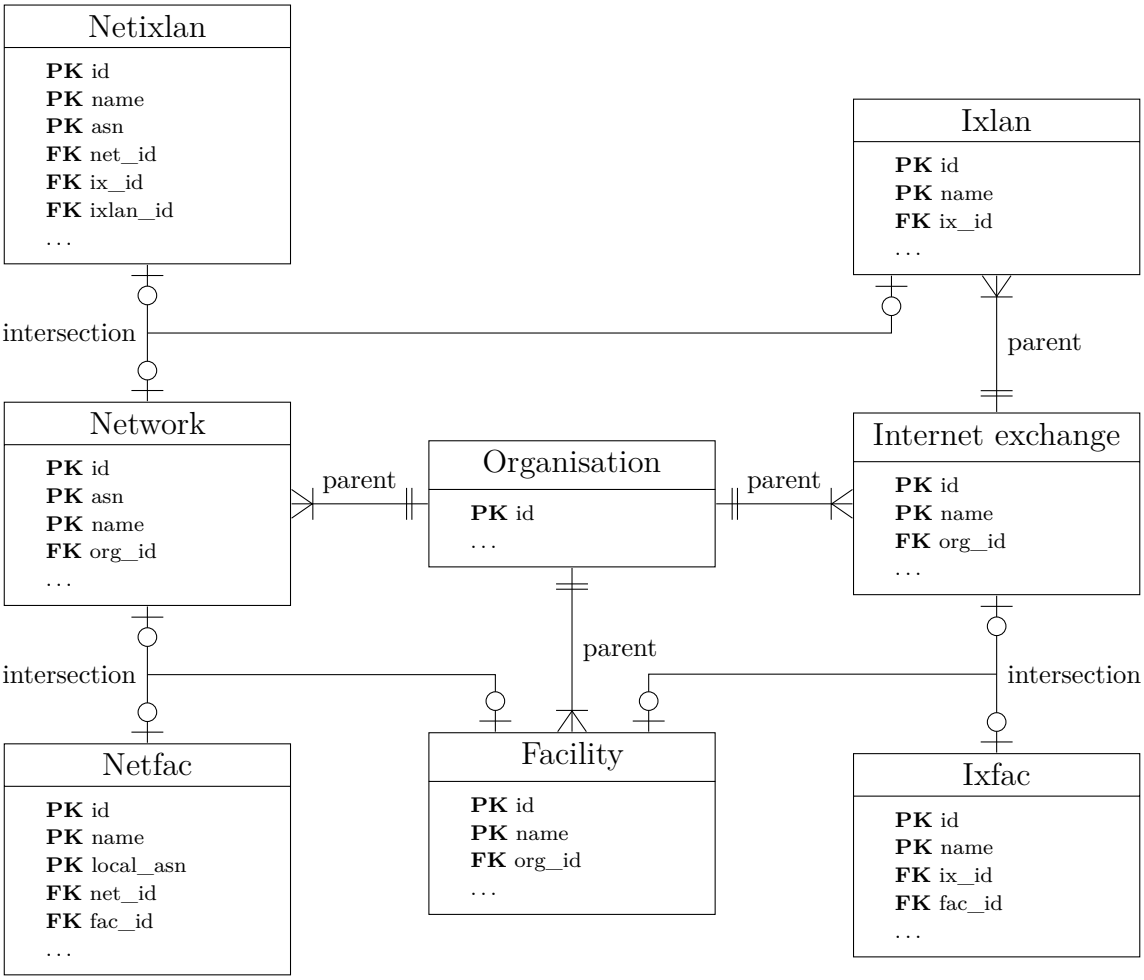


Figure 2.2: Partial representation of PeeringDB's data model.

Organisations These objects are at the top of PeeringDB's object hierarchy. When querying the API[4] for this information, the command returns a list of all registered organisations.

Organisation type objects have many attributes, we are primarily interested in the following selection[5]:

- *id* : non-null and unique identifier for each organisation;
- *name* : should be non-null;

[4]Information on how to query PeeringDB's API can be found in their documentation, at this link : https://www.peeringdb.com/apidocs. For example, in order to query the database for all organisations, the following call is made: `https://www.peeringdb.com/api/org?depth=2`

[5]An exhaustive list of all attributes for all object types can be found in the API documentation (https://www.peeringdb.com/apidocs).

- *net_set* : contains *net* type objects owned by this organisation, may be empty;
- *fac_set* : contains *fac* type objects owned by this organisation, may be empty;
- *ix_set* : contains *ix* type objects owned by this organisation, may be empty;
- *country* : country code for this organisation's headquarter's location;
- *latitude* : the headquarter's latitude;
- *longitude* : the headquarter's longitude;
- *updated* : date and time of the last update to this organisation object (format: "yyyy-mm-ddThh:mm:ssZ").

Facilities Child entity of organisations, a facility can be a data center or a suite within a data center, it is here that networks connect with one another. Organisations lease space inside these buildings to act as points of presence in their network.

As for the organisation objects, the following list of attributes a facility can (or should) have is non-exhaustive:

- *id* : non-null and unique identifier for each facility;
- *org_id* : identifier corresponding to the organisation owning this facility;
- *org_name* : name of the parent organisation;
- *org* : the parent organisation object;
- *name* : the facility's name;
- *net_count* : the number of networks present at this facility;
- *ix_count* : the number of exchanges at this facility;
- *region_continent* : the geographical region this facility is located at (can be Europe, Asia Pacific, North America, South America, Australia, Africa or Middle East);
- *country* : country code where this facility is located;
- *latitude* : latitude from this object's goegraphical coordinates;
- *longitude* : longitude from this object's geographical coordinates;
- *updated* : date and time of the last update to this facility object.

Networks PeeringDB contains detailed data on the characteristics of each network, such as its Autonomous System Number (ASN), network policies, and Points of Presence (PoPs). This information enables network operators to identify potential peering partners, negotiate agreements, and enhance the efficiency of data exchange across the internet infrastructure.

A network type object should have the following attributes:

- *id* : non-null and unique identifier for each network;
- *name* : should be non-null;

- *asn* : the ASN associated to the network, this can also serve as a unique identifier;
- *org_id* : the ID corresponding to the organisation owning this network;
- *info_traffic* : contains information on the bandwidth of the network;
- *info_scope* : if disclosed, this is information on how far the network reaches (this can be, for example, only in North America, or in Asia Pacific and Europe, or globally etc.);
- *ix_count* : number of exchanges where this network is present;
- *fac_count* : number of facilities where this network is present;
- *updated* : timestamp of the latest update made to this object.

Internet exchanges An Internet Exchange (IX) is a physical infrastructure where Internet Service Providers, content providers, and networks interconnect to exchange traffic. It serves as a central hub for routing data between different networks, allowing them to exchange traffic directly rather than routing through third-party networks.

PeeringDB leverages information about IXs to facilitate peering arrangements between networks. It provides detailed data on the networks present at each IX, their peering policies, and contact information. This allows network operators to efficiently establish peering connections and optimise their network routing.

An internet exchange object is a child object of an organisation object, it should have the following attributes:

- *id* : non-null and unique identifier for each IX;
- *name* : should be non-null;
- *org_id* : the ID corresponding to the organisation owning this IX;
- *region_continent* : the geographical region this Internet Exchange is located in (can be Europe, Asia Pacific, North America, South America, Australia, Africa or Middle East);
- *country* : country code where this IX is located;
- *city* : the city this object is active in;
- *fac_set* : a list containing the facility objects where this IX is present;
- *fac_count* : the amount of facilities in the *fac_set* list;
- *ixlan_set* : a list containing the *ixlan* objects[6] where this IX is present;
- *net_count* : the amount of networks at this IX;
- *updated* : date and time of the last update to this object.

Drawbacks One of the biggest shortcomings of PeeringDB is that its information is entirely dependent on what willing Internet operators have added to the database.

[6]These objects are used in PeeringDB to join a network with an Internet Exchange.

Ideally, none of the attributes for the aforementioned entities are null or empty, and all data are up to date. None of these is guaranteed however.

2.2 CAIDA

The "Center for Applied Internet Data Analysis", based at the University of California's San Diego Supercomputer Center, engages in network research and develops research infrastructure to facilitate data collection, organisation, and distribution to scientific researchers [11]. Their mission is to study the Internet's evolution, how it works and behaves, and the way it is put to use. They wish to create an environment where researchers can share and acquire data in order to improve the integrity of Internet science research.

Every day, all of the content available on PeeringDB is queried from the database and put into a JSON file which is then made available to download on CAIDA's website[7].

We have depicted an excerpt of the JSON file in Figure 2.3a, showing what an *ixlan* object looks like in it. The file being too large to format properly, we have added a part that illustrates what it would look like if it were formatted correctly (Fig. 2.3b).

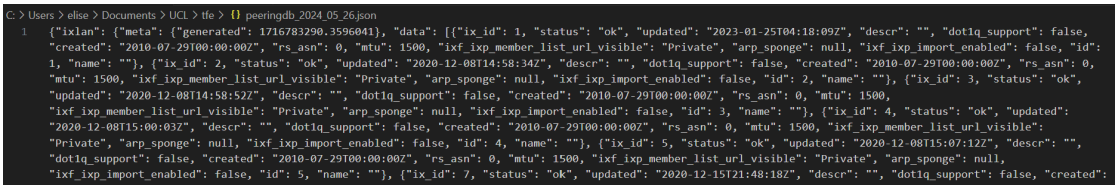
The structure of the JSON is rather intricate and ressembles a dictionary of dictionaries containing lists of dictionaries.

If we compare this to what we know should be present in an *ixlan* object as advertised in PeeringDB's documentation (Fig. 2.3c), we realise there is a level of nested information missing. The absence of these nested sets, i.e. the *ix*, *net_set*, and *ixpfx_set* objects we see on this figure, is caused by a "shallow" query: the depth parameter either wasn't included in the query, or it was set too low to fetch the data we wish to see.

Drawbacks CAIDA's JSON being a daily extract of PeeringDB, the shortcommings of PeeringDB also apply: there is no guarantee the data are complete and up to date. However, there is a far more important drawback, namely the JSON file lacks depth: nested objects that are of interest to us are not represented in the file.

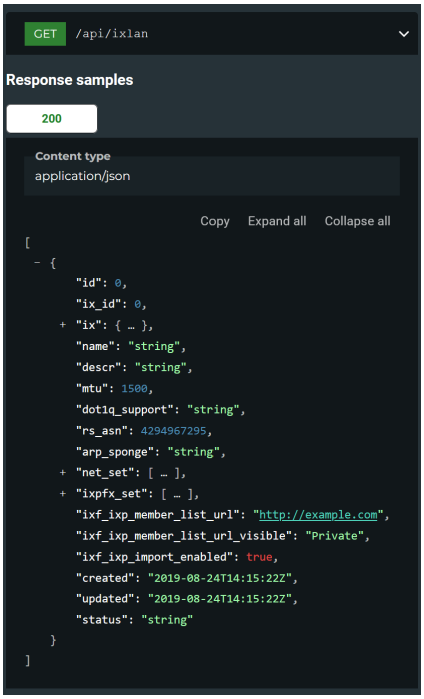
Given these drawbacks of CAIDA's JSON file, PeeringDB remains a valid source for ad-hoc consultation.

[7] https://publicdata.caida.org/datasets/peeringdb/v2/



(a) The fetched JSON file.

(b) Formatted excerpt.



(c) Object attributes as shown in the API.

2.3 Websites of Network Operators and Internet Exchanges

As stated previously, the information made available on PeeringDB depends entirely on the Internet operators willingly providing it. This is why consulting an operator's or Internet Exchange's official website may help gaining more information about their network and presence in facilities, peering with other organisations, etc.

These websites serve as comprehensive resources for potential customers, current subscribers, and stakeholders to learn about the operator's services, infrastructure, and corporate information. We will be using them to collect information on their network coverage, technologies and infrastructures. Operators are, of course, free to display information however they see fit. This means some will have no information at all about their network, some will have a section of plain text stating their

methods (the type of network, its scope, bandwidth, etc) and reachability, while others might have a map of some kind.

When examining a website containing a map (as will be discussed further in Section 4.2.1), it becomes evident that maps are stored in various formats, including bitmaps, PNG images, and rendered JSON files. Furthermore, the scope of information conveyed through these maps varies significantly among operators. While some operators merely pinpoint their own facilities, others include datacenters where they lease space. Some operators provide more details on the routing between facilities, but the level of detail varies much. In some cases there is simply a direct line between facilities; in other cases way points along cable routes are indicated; and in rare cases there may be more detail when paths are linked to major public infrastructures like highways and railways.

Drawbacks The lack of a standardised data format, structure and level of granularity makes it difficult to extract structured data and analyse it.

Furthermore, there is no guarantee that the information is complete, accurate and up-to-date.

2.4 Example

By way of example let's see what information the various sources offer on one of the Internet Exchanges, namely Amsterdam Internet Exchange (AMS-IX).

AMS-IX [12] is a "neutral member-based association that operates multiple interconnection platforms around the world". The leading platform is based in Amsterdam; it is one of the longest hubs for internet traffic in the world. As a matter of fact, AMS-IX has developped its own fiber-optic Metropolitan Area Network (MAN) to enhance local connectivity between its various Points of Presence (PoP) in the area (see Figure 2.4).

Figure 2.5 shows what information we find on the website concerning the PoPs, and Figure 2.6 shows the information retrieved from PeeringDB.

From these images, we can conclude that this is a case where both sources provide the same information, which raises our confidence in the contents of PeeringDB.

2.5 Data Enrichment

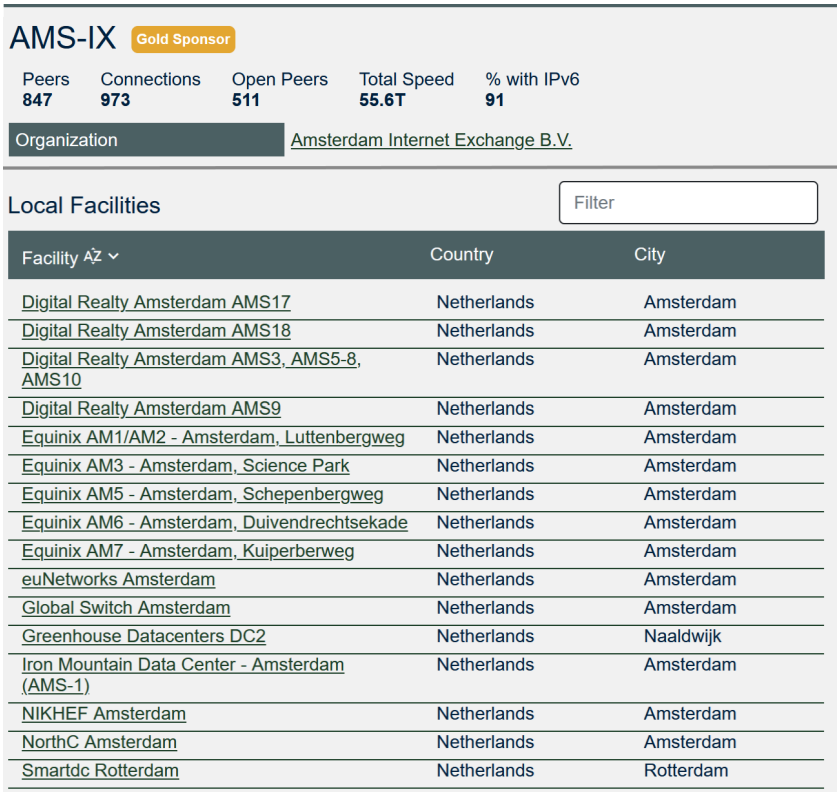
From the preceding sections, it becomes clear that neither source of information can be trusted to be complete, error-free, and up to date.

Figure 2.4: AMS-IX data centers connected in a MAN.

1	NorthC Amsterdam (AMS01)	+31 (0)20 486 9773	Kabelweg 48a, Amsterdam	VISIT WEBSITE
2	Digital Realty AMS17	-	Science Park 120, Amsterdam	VISIT WEBSITE
3	Digital Realty AMS18	+31 (0)20 480 4415	H.J.E. Wenckebachweg 127, Amsterdam	VISIT WEBSITE
4	Equinix AM1/2	+31 (0)53 434 0570	Luttenbergweg 4, Amsterdam	VISIT WEBSITE
5	Equinix AM3	+31 (0)20 808 0015	Science Park 610, Amsterdam	VISIT WEBSITE
6	Equinix AM5	+31 (0)20 592 8263	Schepenbergweg 42, Amsterdam	VISIT WEBSITE
7	Equinix AM6	+31 (0)53 436 2666	Duivendrechtsekade 80A, Amsterdam	VISIT WEBSITE
8	Equinix AM7	+31 (0)53 434 0570	Kuiperbergweg 13, Amsterdam	VISIT WEBSITE
9	euNetworks	+31 (0)20 354 8098	Paul van Vlissingenstraat 16, Amsterdam	VISIT WEBSITE
10	Iron Mountain	+31 (0)20 316 5170	J.W. Lucasweg 35, Haarlem	VISIT WEBSITE
11	Global Switch	+31 (0)20 666 6300	Henk Sneevlietweg 2-6, Amsterdam	VISIT WEBSITE
12	Digital Realty AMS5	+31 (0)20 880 7700	Tupolevlaan 101, Schiphol-Rijk	VISIT WEBSITE
13	Digital Realty AMS9	+31 (0)20 560 6600	Science Park 121, Amsterdam	VISIT WEBSITE
14	Nikhef	+31 (0)20 592 2037	Science Park 105, Amsterdam	VISIT WEBSITE
15	Greenhouse	+31 (0)17 420 0222	Industriestraat 53, Naaldwijk	VISIT WEBSITE
16	Smartdc	+31 (0)10 890 0048	Van Nelleweg, Rotterdam	VISIT WEBSITE

Figure 2.5: AMS-IX PoPs on website.

For the purposes of this thesis, we feel that a repeated consultation of the various sources and crossing of information is the best path forward. This will be



AMS-IX Gold Sponsor

Peers	Connections	Open Peers	Total Speed	% with IPv6
847	973	511	55.6T	91

Organization: Amsterdam Internet Exchange B.V.

Local Facilities

Filter

Facility	Country	City
Digital Realty Amsterdam AMS17	Netherlands	Amsterdam
Digital Realty Amsterdam AMS18	Netherlands	Amsterdam
Digital Realty Amsterdam AMS3, AMS5-8, AMS10	Netherlands	Amsterdam
Digital Realty Amsterdam AMS9	Netherlands	Amsterdam
Equinix AM1/AM2 - Amsterdam, Luttenbergweg	Netherlands	Amsterdam
Equinix AM3 - Amsterdam, Science Park	Netherlands	Amsterdam
Equinix AM5 - Amsterdam, Schepenbergweg	Netherlands	Amsterdam
Equinix AM6 - Amsterdam, Duivendrechtsekade	Netherlands	Amsterdam
Equinix AM7 - Amsterdam, Kuiperberweg	Netherlands	Amsterdam
euNetworks Amsterdam	Netherlands	Amsterdam
Global Switch Amsterdam	Netherlands	Amsterdam
Greenhouse Datacenters DC2	Netherlands	Naaldwijk
Iron Mountain Data Center - Amsterdam (AMS-1)	Netherlands	Amsterdam
NIKHEF Amsterdam	Netherlands	Amsterdam
NorthC Amsterdam	Netherlands	Amsterdam
Smartdc Rotterdam	Netherlands	Rotterdam

Figure 2.6: AMS-IX PoPs on PeeringDB.

further elaborated in Chapter 4. Prior to that, we discuss in Chapter 3 the data model we have used to consolidate the data from the various sources discussed in this chapter.

Chapter 3

Data model

We have adressed PeeringDB's structure and the important relevant information it provides in Section 2.1. We have seen that PeeringDB advertises the nodes of the different networks (presence in IXes and other points-of-presence) but does not provide information on how the facilities hosting a same operator are connected with one another.

The operator itself may advertise data on interconnections between facilities on its website. However, that data sometimes lacks precision when it comes to identifying information about the facility such as its exact geocoordinates or name.

We need to reconcile both information sources and the data they provide, namely the detailed information on facilities from PeeringDB and the precisions regarding how they are connected, which we retrieve from operator websites. To do this, we have designed our own data model (which we have illustrated in Figure 3.1) based on the one provided by PeeringDB.

In our database, we have defined tables for the major entities of PeeringDB (i.e. organisations, networks, facilities, and internet exchange points), and we have also defined a few additional junction tables (i.e. *netix*, *ixfac*, and *faclink*) in order to be able to display connections between facilities on our map, and more generally display the paths traversed when packets travel the Internet and hop from AS to AS.

The information collected in this database has continuously evolved throughout the progression of this project and is still subject to future modification. Starting off as a simplified version of PeeringDB's database containing only the four main tables, it has slowly become a source of information in its own right, containing data unavailable in PeeringDB.

In this section, we will be elaborating on how we designed our data model and its differences with the data model from PeeringDB it was based on.

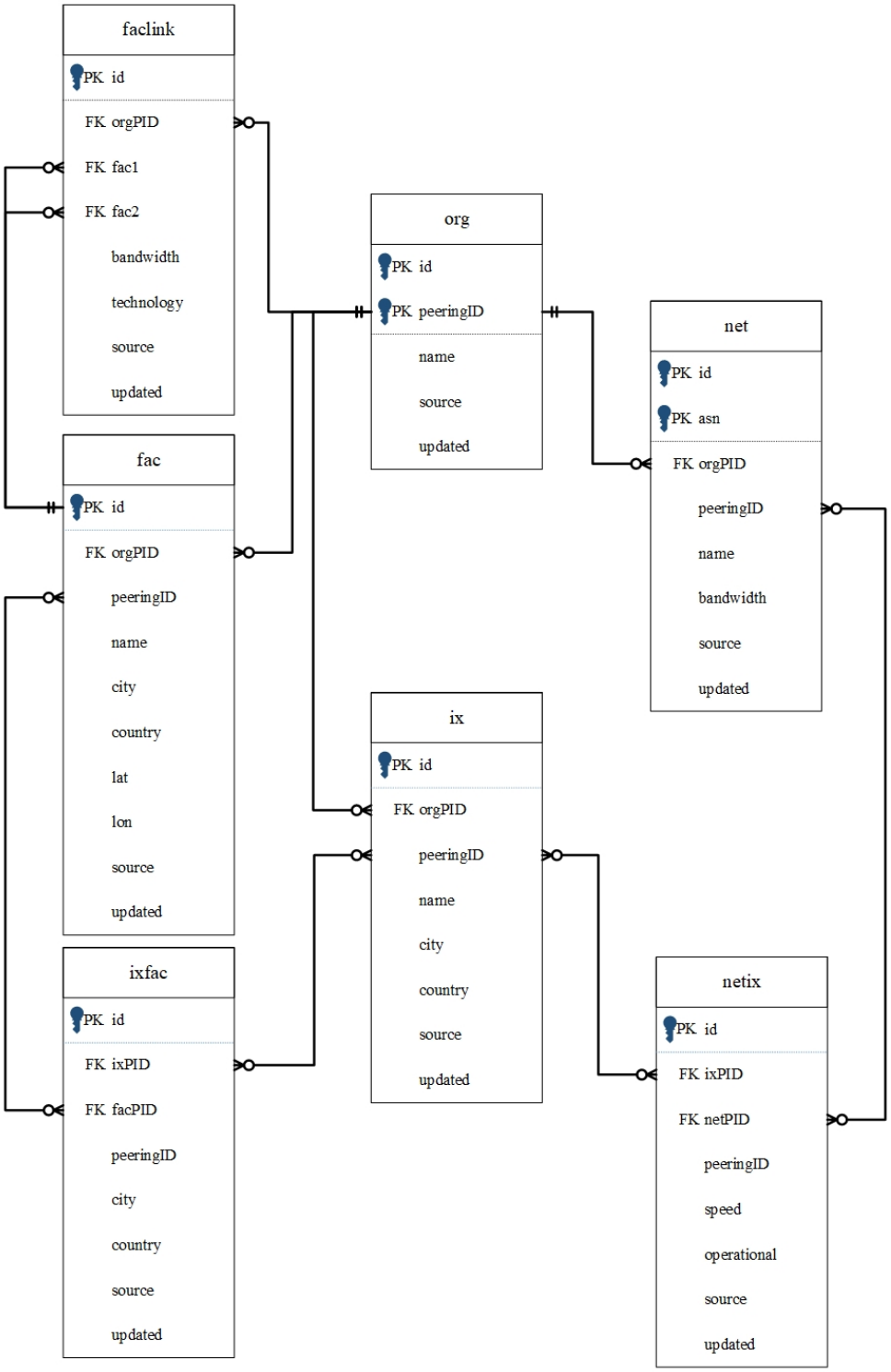


Figure 3.1: Entity relationship diagram of our database.

3.1 Organisations

In order to be able to uniquely identify each organisation that will be stored in our database, we decided to use ID numbers the same way PeeringDB does. However, the possibility of finding operators online that are not advertised on PeeringDB meant we could not rely solely on the *id* attribute organisation objects have (cf. Section 2.1). To compensate for this, we have defined an additional *id* column while still keeping the identifier column from PeeringDB for reidentification purposes in PeeringDB's database. This means we have two ID columns: *id*, which is the auto-incremental column we defined, and *peeringID*, where we store the ID numbers from PeeringDB.

On top of these two ID columns, we have defined the following other attributes:

- *name*: to store the organisation's name, it is the equivalent to the *name* attribute from PeeringDB;
- *source*: meant to indicate where the information was retrieved from, its value is "PeeringDB" by default;
- *updated*: a timestamp of the last update made to the row in this table[1].

An excerpt from the *org* table is shown in Table 3.1.

id	peeringID	name	source	updated
1	2	Equinix, Inc.	peeringDB	2024-05-30 21:16:04
...	...	...	...	...
86	214	BELNET	peeringDB	2024-05-30 21:16:04
...	...	...	...	...

Table 3.1: Representation of two organisations in the database. Underlined columns are defined as primary keys; italic columns are foreign keys; others are simple attributes.

PeeringDB contains more precise information on organisations, but we are only interested in being able to associate facilities, networks and internet exchange points to these organisations (which we achieve through foreign keys pointing to the organisation's *peeringID* attribute). This is what prompted us to discard the other possible attributes.

[1] In a "YYYY-MM-DD HH:MM:SS" format.

3.2 Facilities

The main purpose of a facility type object in our data model is to characterise a data center node we wish to display on the map in order to recreate the physical look of a network.

Facilities are identified by a unique auto-incremental numerical value we assign to them. This *id* attribute follows the same logic as explained in Section 3.1: to avoid conflicts with future updates in PeeringDB[2], we need to create a new *id* attribute that can accompany the one assigned by PeeringDB.

Since we wish to display facilities on a map as nodes, the objects need geographical coordinates. We store these in the *lat*, *lon*, *city*, and *country* columns. We also added a *source* attribute in order to keep track of the origin of the information. This means rows in the *fac* table will look like the excerpt in Table 3.2. We can see

id	*orgPID*	peeringID	name	city
1	31	18	NIKHEF Amsterdam	Amsterdam
...	...	...	...	...
2432	NULL	NULL	Rome35836	Rome
...	...	...	...	...

country	lat	lon	source	updated
NL	52.356394	4.950837	peeringDB	2024-05-27 22:32:26
...	...	...	...	...
IT	41.900000	12.483300	tisparkle	2024-05-28 22:57:45
...	...	...	...	...

Table 3.2: Representation of two facilities in the database. Underlined columns are defined as primary keys; italic columns are foreign keys; others are simple attributes.

in the table above that information we retrieved from operators is not necessarily as complete as information parsed from PeeringDB. In Chapter 4, we further develop how information is obtained, but note that we might find `NULL` values in:

- The *orgPID* attribute, which comes from not being able to identify which organisation owns the facility. Unless the facility's name contains the name of the organisation that owns it (as is the case for e.g. *Colt*'s or *Verizon*'s facilities[3]), it is troublesome to find which organisation owns the facility.

[2]If we use the ID from PeeringDB as identifier and assign new ones to the facility objects we fetch from other sources, this might cause conflicts if those identifiers are used in future updates in PeeringDB.

[3]This can be observed on the respective PeeringDB pages of these organisations.

- The *peeringID* attribute, caused by the absence of the facility in PeeringDB, and thus it not having an *id* in their database.
- The *country* attribute, which is due to the information not being automatically parsable from the operator's information. If we wanted to add a country to this entry in the table, we would need to locate the facility on the map from the ISP's website and identify its country before manually modifying the entry in the database. The same logic applies for the *city* attribute.

Facility objects have a relationship with IX objects, which is documented in the *ixfac* table (Section 3.6), and facilities can have relationships with one another: these are represented by the *faclink* table (Section 3.7).

3.3 Networks

The network table of our data model mainly has a supporting role for derived objects. It allows us to identify an ASN and what organisation it belongs to. Similarly as for the tables discussed previously, a *net* object is primarily identified by an ID assigned by us, along with its ASN, and if applicable an ID from PeeringDB's database. Table 3.3 gives an excerpt.

id	asn	*orgPID*	peeringID
1	4436	8897	1
...	...	...	...
3379	132405	9646	7024
...	...	...	...

name	bandwidth	source	updated
GTT Communications (AS4436)	Not disclosed	peeringDB	2024-05-30 21:16:09
...	...	...	...
Summit Internet	1-5Gbps	peeringDB	2024-05-30 21:16:09
...	...	...	...

Table 3.3: Representation of two networks in the database. Underlined columns are defined as primary keys; italic columns are foreign keys; others are simple attributes.

We wish to establish a relationship between internet exchange and network objects in order to identify which ASNs are present at which IXes. As such, they are parent objects to the *netix* table discussed in Section 3.5.

3.4 Internet Exchanges

This table documents the different Internet Exchange (IX) objects across Europe. It is a key component of the data model as it ultimately will allow us to identify how ASNs connect from one place to another over the Internet. We have defined the following attributes:

- *id*: the primary key, an integer value we assign to each row in order to allow future additions from information sources other than PeeringDB;
- *orgPID*: corresponds to the PeeringDB ID of the organisation that owns this IX;
- *peeringID*: the ID assigned to the object in PeeringDB's database;
- *name*;
- *city* and *country*: the city and country the IX is located in;
- *source* and *updated*: same principle as for all the tables in our database, the source is meant to keep a trace of the listed information's origin and *updated* is the timestamp at which it was last modified inside our DB.

Table 3.4 shows an excerpt.

id	*orgPID*	peeringID	name
1	791	18	LINX LON1
...	...	...	...
280	3207	3290	MIX Palermo
...	...	...	...

city	country	source	updated
London	GB	peeringDB	2024-05-30 21:16:14
...	...	...	...
Palermo	IT	peeringDB	2024-05-30 21:16:14
...	...	...	...

Table 3.4: Representation of two IXes in the database. Underlined columns are defined as primary keys; italic columns are foreign keys; others are simple attributes.

3.5 Netix

Netix objects are part of the junction entity between networks and internet exchange points. They are equivalent to the *netixlan* entities in PeeringDB. The attributes defined in this table are based on the ones defined in the parent entities.

- The *ixPID* and *netPID* attributes correspond to the PeeringDB ID associated to the IX and network, respectively, concerned by the relationship represented by this table,
- The *speed* and *operational* attributes representing the networking speed and availability of this peer are taken directly from the *netixlan* object.

Table 3.5 shows an excerpt.

id	*ixPID*	*netPID*	peeringID
1	73	2	68
...	...	...	...
1400	291	942	11601
...	...	...	...

speed	operational	source	updated
100000	1	peeringDB	2024-05-30 23:12:17
...	...	...	...
10000	0	peeringDB	2024-05-30 23:12:18
...	...	...	...

Table 3.5: Representation of two network-ix relationships in the database. Underlined columns are defined as primary keys; italic columns are foreign keys; others are simple attributes.

It is important to note that we interpret the presence of an ASN at an exchange point to mean it can connect to any other ASN present at this point. This is not necessarily the case in reality, but there is no way for us to know which ASNs are effectively in contact based on the information we fetch from PeeringDB.

3.6 Ixfac

The *ixfac* table is comparable in function to the *netix* table discussed previously. Ixfac objects are part of the junction entity between facilities and Internet Exchangees. This means the objects documented in this table are facilities in which a certain IX is present, and conversely, the IXes present at a certain facility. The attributes we chose to characterise this object type are the following.

- As for all previous objects, an *id* and *peeringID* to uniquely identify the object, as well as the *source* and *updated* columns.
- *ixPID* and *facPID*, linking back to the objects concerned by the relationship.

- *city* and *country*, which serve the purpose of locating where the facility and IX meet.

Table 3.6 contains an excerpt.

id	*ixPID*	*facPID*	peeringID
1	18	40	28
...	...	...	...
220	358	837	684
...	...	...	...

city	country	source	updated
London	GB	peeringDB	2024-05-31 14:00:24
...	...	...	...
St. Petersburg	RU	peeringDB	2024-05-31 14:00:25
...	...	...	...

Table 3.6: Representation of two facility-ix relationships in the database. Underlined columns are defined as primary keys; italic columns are foreign keys; others are simple attributes.

3.7 Faclink

The *faclink* table was created to reproduce the network maps advertised by Internet operators on their website. The table needs to be populated manually, as it requires us to verify which data centers are connected (this is discussed further in Section 4.2.1). It has the following attributes.

- *id*: the primary key, an integer value we assign to each row.
- *orgPID*: the PeeringDB ID number corresponding to the organisation that owns this facility connection.
- *fac1* and *fac2*: the two facilities to be connected. Each refers to the PeeringDB ID of one of the facilities on this link.
- *bandwidth*: if the information is obtainable, either on PeeringDB or on the organisation's website, we assign to this attribute the bandwidth available between two facilities.
- *technology*: indicates whether the facilities are connected through fiber-optic cables, copper, satellite connections, etc..
- *source* and *updated*: the attributes used to keep track of the information source and latest update.

Table 3.7 shows an excerpt.

id	*orgPID*	*fac1*	*fac2*
1	17474	17	2374
...	...	...	...
111	78	2445	2556
...	...	...	...

bandwidth	technology	source	updated
100-200 Gbps	Fibres	hawetelekom	2024-05-28 19:20:29
...	...	...	...
10-20 Tbps	NULL	tisparkle	2024-05-29 06:21:08
...	...	...	...

Table 3.7: Representation of a relationship between two facilities in the database. Underlined columns are defined as primary keys; italic columns are foreign keys; others are simple attributes.

Chapter 4

Data Collection and Consolidation

In Chapter 3, we designed the structure of the database. In order to allow research to happen, we need actual data. In this chapter, we explain how this data is collected and consolidated.

4.1 Initial Data Load

Loading data into a database for the first time is commonly referred to as the *initial data load*. It typically involves the process of transferring data from one place (another database, a file, or an external system like an API) to another by inserting data into tables and defining relationships between them to form the target database.

This section is divided into three parts defining the different steps towards completing the initial load of our database. This three-step process is commonly referred to as an ETL operation: Extract, Transform, Load.

4.1.1 Extract

We want to get our initial load from PeeringDB's database in order to get a collection of the facilities across Europe, which we consider as the nodes in a graph.

Fetching PeeringDB's information can be done through two methods, as discussed in Chapter 2: (1) directly from PeeringDB's API, or (2) through CAIDA's JSON files. We initially worked with the latter option, but due to the "shallowness" of CAIDA's JSON file, we ultimately decided to extract the data by repeatedly calling PeeringDB's API for each object type that is of interest to us, i.e. *ixfac*, *netixlan*, *ix*, *net*, *fac*, and *org*.

4.1.2 Transform

For each object type in PeeringDB, we launch a query on its API. This operation results in a dictionary containing two keys: 'meta' and 'data'. We are only interested in the second one.

This data is a list of dictionaries, each one representing an element of the object type we are querying for. The list elements hold a long series of key-value pairs pertaining to the instance

At this point the lists contain all instances present in PeeringDB of the corresponding entity type, i.e. the lists have a worldwide coverage. As we are only interested in European networks, we perform a cleaning operation based on continent or country code, only maintaining data regarding European networks (see Figure 4.1).

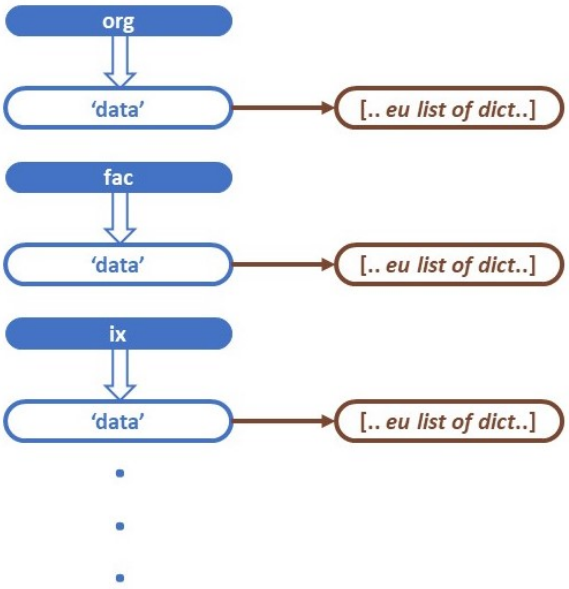


Figure 4.1: Lists of European data.

In the final step of the parsing process, we are only interested in the organisations, facilities, internet exchange points, and networks. We construct lists of simplified dictionaries for each of these entity types (an example is shown in Figure 4.2). The dictionaries are simplified as we eliminate impertinent keys from the earlier dictionaries (e.g. we remove social media information such as operators' websites).

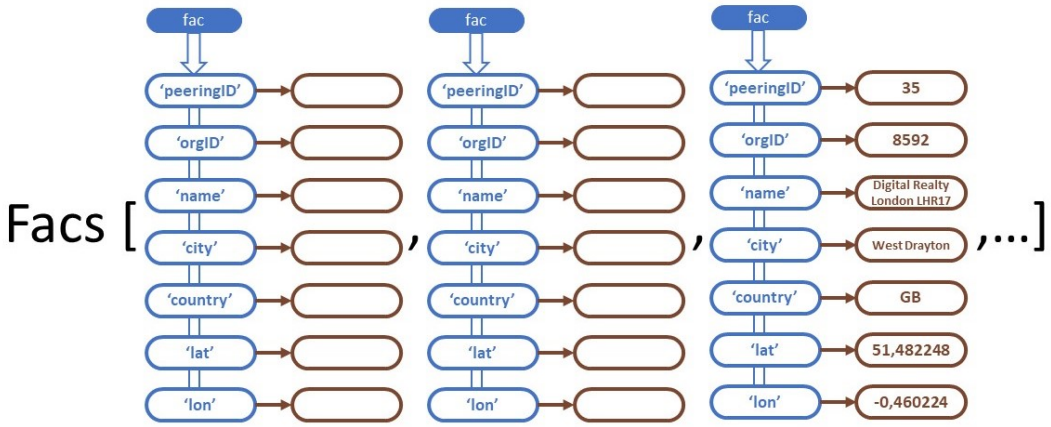


Figure 4.2: Visualisation of the list of simplified dictionaries.

4.1.3 Load

To load the information, we had to chose what type of database to use: object oriented databases, NoSQL databases (key-value stores, graphs, ...), or relational databases (MySQL, Postgres, ...).

Choice of database technology

Object oriented databases incorporate principles of object-oriented programming directly into database systems. This integration results in a smoother compatibility with the object oriented data structures commonly employed in application development.

These databases are can store complex data structures such as objects (which can include other objects, arrays, and even multimedia types directly). They support the concept of inheritance from object-oriented programming, allowing objects to inherit properties and methods from other objects. This feature enables more efficient data modeling and reduces redundancy. This type of database excels in scenarios where applications need direct manipulation of intricate data structures, but they may not be the best choice for applications where extensive querying is necessary or where the data naturally conforms to a tabular format [13].

Considering our data necessitates a lot of querying and does not require intricate data manipulation methods and inheritance of such methods of such objects, we found this database type ill-suited for the purpose of our application.

NoSQL databases, alternatively known as non-relational databases, employ storage models tailored to the unique needs of the stored data. This might entail

storing data as key-value pairs, JSON documents, or even as a graph structure made up of edges and vertices.

These databases handle vast amounts of unstructured data, and offer flexibility to accommodate evolving business needs. NoSQL is the best selection for flexible data storage with little to no structure limitations [14]. Looking at our application (which is highly structured, as discussed in Chapter 3), this type of database is not suited for our purposes.

This type of data stores doesn't use SQL, hence its name *NoSQL*, instead other programming languages and constructs to query the data are employed. As we intend to use SQL, this is a further reason not to use this type of database.

Relational databases contain organised data with predefined relationships, stored within one or more tables or "relations", consisting of columns and rows. This structure facilitates clear visualisation and comprehension of the interconnections between various data elements.

These databases feature structured data with enforced relationships. They boast extensive indexing capabilities, resulting in swift query responses. Relational databases are particularly well-suited for demanding transactional applications requiring robust data handling. [14].

For the purposes of building the solution we have in mind, a relational database best suits our goals. Amidst the various possibilities of relational database management systems, we chose to work with a MySQL database because of our greater familiarity with this type of database.

Loading of the database

Now that we have determined what kind of database to work with, we still need to populate it with the information we have extracted from PeeringDB and transformed to suit our data model.

This is achieved by launching a Python program that executes SQL queries based on a set of parameters, namely: one of the predefined lists we stored the parsed information in, the name of the corresponding table in the database, the number of columns the table has, and the name of our database. With this information, our function can execute a series of SQL `INSERT` queries after which the table will contain the information we parsed for its corresponding object type. This operation thus needs to be repeated for the organisations, facilities, networks, internet exchanges, *ixfac*s, and *netix*es.

A visualisation of the process of transferring information from our dictionaries to the database can be seen in Figure 4.3, specifically for the facility object type.

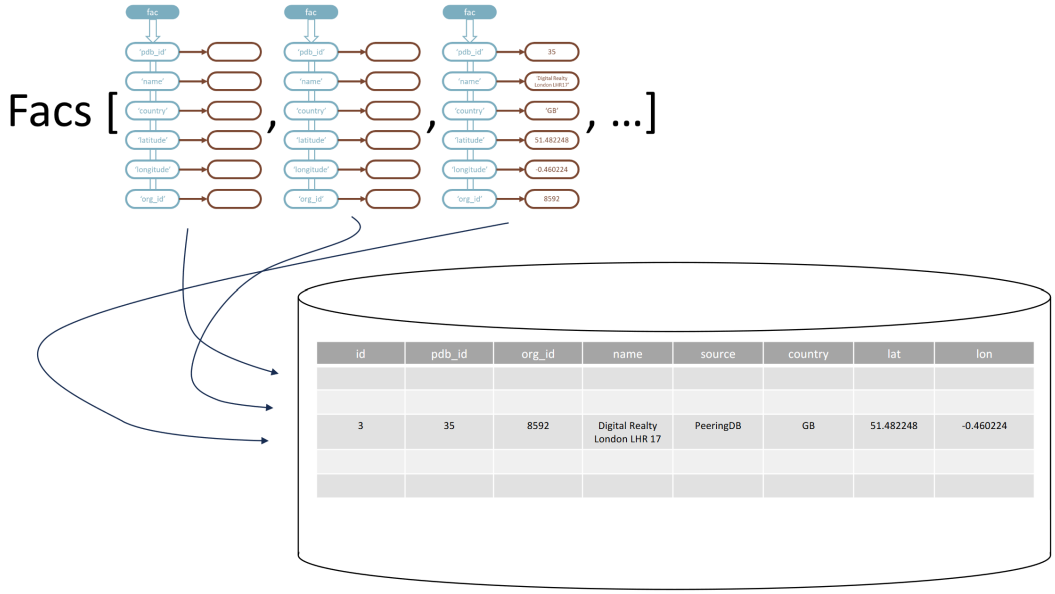


Figure 4.3: Loading an object of facility type in the database.

4.2 Database Enrichment

After we have performed the initial data load, our database contains a certain amount of information, and we now wish to start generating a map. The issue is that we still do not have data for the *link* table. This comes from the absence of information available on the details of facility interconnections in PeeringDB.

So before anything else, we will need to find a solution to this information gap. We do this by consulting ISPs' official websites and searching for their network coverage maps (as discussed in Section 2.3).

4.2.1 Enrichment

When comparing the information derived from an ISP's website with the data obtained from PeeringDB, we may notice that the latter's information is incomplete (e.g. an operator that does not register all of its ASes in PeeringDB, or alternatively, an ISP that isn't present in PeeringDB; data centers that aren't present in the initial data load; attributes with missing values; etc.).

In order to resolve these issues, we have devised the following protocol. We start by selecting the ASN present at the largest amount of facilities[1], and assuming

[1]Finding which network is the most prevalent in facilities can be done by looking at the *fac_count* attribute of the network object.

this is the network with the most coverage (and, by extension, the one already containing the most information).

With the help of that ASN's coverage map, from the operator's website, we can determine what facilities are not registered in PeeringDB; which ones have become inactive; we can see what data centers are connected; ... However, the format in which this map is displayed heavily impacts our protocol.

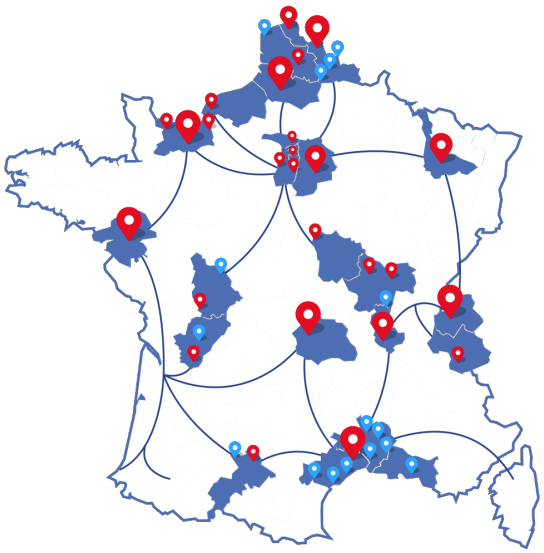


Figure 4.4: Covage coverage map (image from [15]).

Indeed, there are many different ways to generate a map on a webpage. In the worst case, an operator doesn't display any map at all, or it displays one with such low resolution that it is near impossible to interpret the data sufficiently accurately for this project (see for example, Covage[2], where the map displays the location of data centers with too little detail to be interpretable as can be seen in Figure 4.4). In this situation, it is best to directly contact the operator and enquire about its network[3].

Another possible scenario is when the map is just detailed enough to interpret, or it is generated based on a fetchable and external information source (see, for example, Colt's coverage map[4] in Figure 4.5, where points are drawn on the map based on coordinates found in a remote file).

Best case, the map is interactive and we can click on nodes to obtain more information. These maps are usually built with data that can be retrieved through HTTP `GET` requests (used to fetch data from a server at the designated resource).

[2] https://www.covage.com/reseaux/

[3] We have initiated several enquiries, but none have been responded til date.

[4] https://www.colt.net/global-network/coverage-maps/

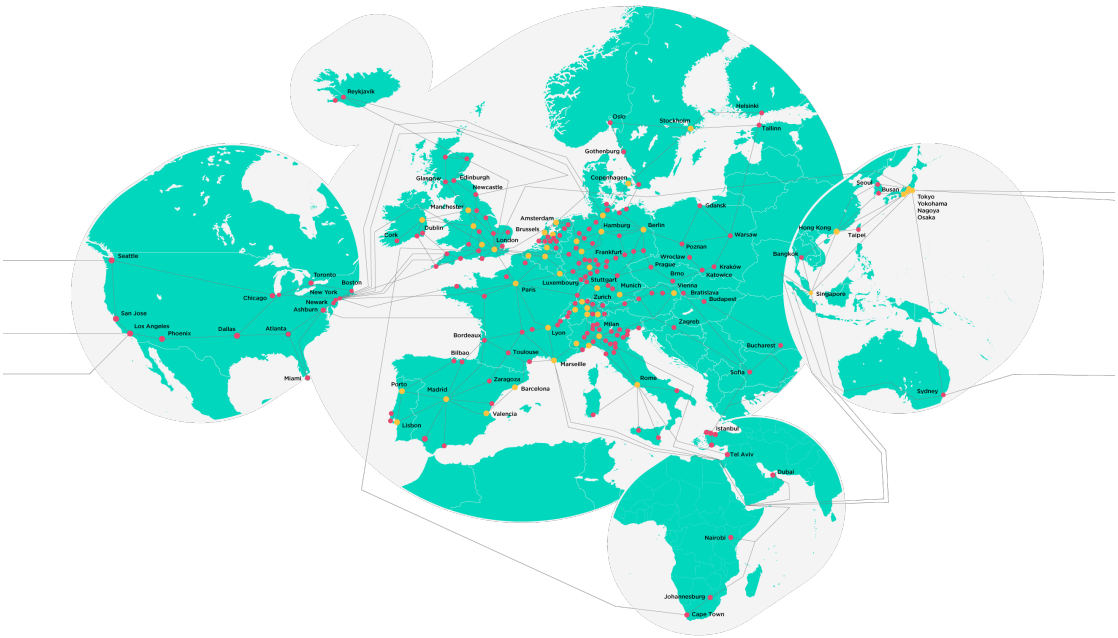


Figure 4.5: Colt coverage map (image from [16]).

These last two options are the ones of greatest interest to us. They allow us to fetch the data through the `GET` requests ourselves; which we can then parse and add to our database. Parsing the response of the executed requests still requires us to individually analyse and tailor the retrieval of information to each response as every response will be different based on how the information is stored by the operator.

Adding the data retrieved from the ISPs' websites to our database allows us to ensure its completeness, in the sense all nodes of a facility network will now be present. We need to be careful though not to add any duplicate information (we consider two facilities to be duplicates if they are situated within a 200 meter radius from one another, and verify this by applying the haversine formula[5]). Note that not all request responses provide a value for every attribute in our database, this is why we may still have `NULL` values. Again, this may be remedied by contacting the operator.

[5]Formula to find the distance between two points on a sphere, using their latitudes and longitudes.

4.2.2 Creating Links

The procedure we follow to add connections in our database is: (1) observe connections on an ISP's coverage map; (2) identify the facilities concerned by this connection; and (3) add a row with the corresponding information to the *faclink* table.

Let's illustrate these steps through an example. On the figure below (Figure 4.6), we have illustrated the European portion of the Terrestrial Backbone network from Telecom Italia Sparkle[6]. The first step is to observe the many different connections between the many different data centers (represented by nodes in the graph). In doing this, we can recognise connections between cities such as the ones linking Paris to London, Stockholm to Berlin, Milan to Rome, etc.

Figure 4.6: Coverage map from Telecom Italia Sparkle.

The advantage of this map is its interactivity: when clicking on one of the nodes, one gets additional information on the data center(s) in that city. This helps in identifying exactly which facilities correspond in our database, allowing us to document this connection with the usage of the facilities' IDs (see a link object's data model in Section 3.7).

[6] https://www.globalbackbone.tisparkle.com/

Take, for instance, the link between Turin (Italy) and Lyon (France) as shown in Figure 4.7.

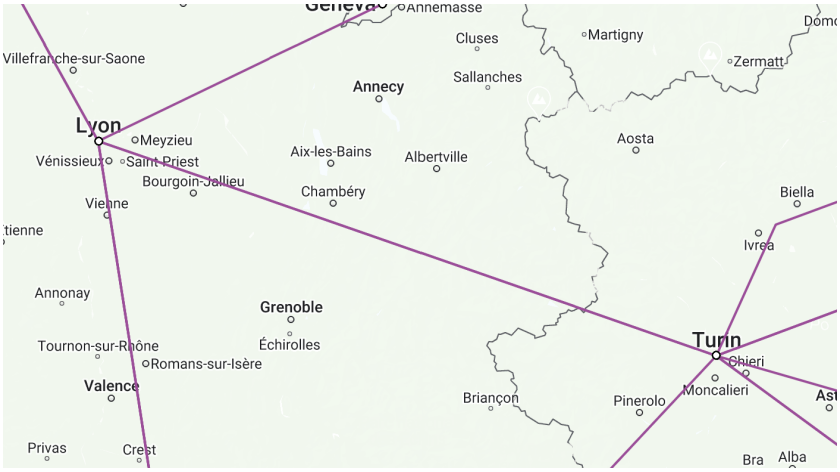


Figure 4.7: TISparkle connection between Lyon and Turin.

We are interested in knowing which facility in each of the cities is a node on this link. We find the information in Figures 4.8a and 4.8b. Lyon only lists one Point of Presence (PoP), thus we can easily identify this facility within the database (see Table 4.1) and associate it to the node at the one end of the link. Turin lists multiple PoPs; by default – and for the sake of simplicity –, we choose the facility with the smallest ID, resulting in the second listed facility being chosen as representative data center of the Turin node (see Table 4.2).

id	*orgPID*	peeringID	name	city
15	11780	57	SFR Netcenter Lyon-Venissieux	Vénissieux

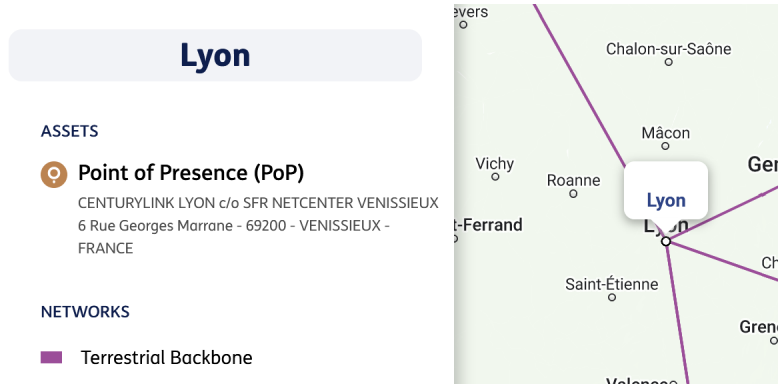
country	lat	lon	source	updated
FR	45.722882	4.862905	peeringDB	2024-05-30 21:16:08

Table 4.1: The Lyon facility in the database.

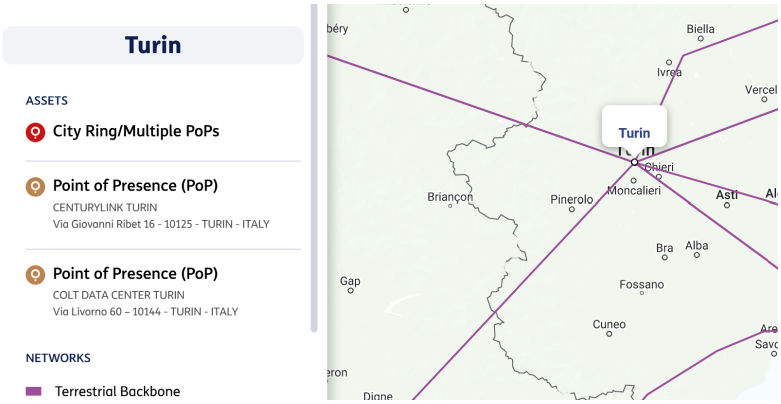
id	*orgPID*	peeringID	name	city
1025	118	5887	COLT Datacenter Turin	Turin

country	lat	lon	source	updated
IT	45.088155	7.671782	peeringDB	2024-05-30 21:16:09

Table 4.2: The Turin facility in the database.



(a) Facility in Lyon.



(b) Facilities in Turin.

Figure 4.8: Data centers in Lyon and Turin.

Having found what data centers to use as endpoints, we can now add a row in our *faclink* table corresponding to this connection. This is done by launching an SQL `INSERT INTO` query. Since we had to perform this task many times, we wrote a Python function for the purpose of inserting a link into the database. It takes as parameter a dictionary where the key-value pairs are the column name and corresponding value for each link. In the case of our example, the dictionary would look like this: `{"orgPID":78, "fac1":"15", "fac2":"1025", "bandwidth":"10-20Tbps", "technology":NULL, "source":"tisparkle"}`[7].

The resulting row after adding this link to the database can be seen in Table 4.3.

[7]The *orgPID* and *bandwidth* information were retrieved by looking up TISparkle on PeeringDB's user interface.

id	*orgPID*	peeringID	name	city
1025	118	5887	COLT Datacenter Turin	Turin

country	lat	lon	source	updated
IT	45.088155	7.671782	peeringDB	2024-05-30 21:16:09

Table 4.3: The Lyon-Turin link in the database.

4.3 Updating

Once the initial load has been completed, the matter of keeping the database up to date arises.

We consider two ways of adressing the matter: we perform updates periodically (once a month, for example, because of the 'slowly varying' character of our data), and whenever the database is needed/used in the context of a study (in which case we should be concerned only about the operators of interest to the study).

PeeringDB objects have *updated* attributes to indicate when the object was last updated. Our database's tables have this column as well. Updating an element should thus be as simple as verifying the concerned objects timestamps are more recent.

Chapter 5

Data Representation

We have collected all available data from the various sources, we have built and populated our database, and we have made sure the DB contains the latest updates; now, we want to visualise its contents. We wish to display this in two maps: one to show connections between facilities, and one to draw the shortest paths taken by ASNs to travel from one IX to another.

We will start by having a look at the different ways of doing this in Python.

5.1 Connecting to the Database

In order to generate a map based on the database we created, we need to connect to it. We will then be able to launch queries and precisely select the information we need from our tables.

This connection is established with the help of the PySQL package, which allows us to connect to the database through configuration files. Code used to perform queries on the database[1] was inspired from tutorial sites [17].

5.2 Generating a Map

Before all else, we need to determine a few requirements in order to choose the best way to proceed.

1. If we use a static environment to display the elements of our map, they will be too condensed and make the resulting map unreadable. To counter this, we can use an interactive environment.

[1]All code written for this project can be found at the following repository address: https://forge.uclouvain.be/peiffere/tfe

2. If we wish to identify the nodes on the map, we need to be able to differentiate them. We can do this by making nodes clickable, which would then display some information that will allow us to determine which row corresponds in the database.
3. Another useful characteristic of the map is to congregate nodes to have a cleaner display.
4. A map can display many different links and nodes, and it can get confusing if there is no way of differentiating what each link/node stands for. In the case of the nodes, it can be a data center or an internet exchange point; for the links we can have connections between facilities or the path of an ASN between IXes; future developments might mean additional node and link types we will also need to be able to tell apart. Possible solutions to this issue include using colors or icons and creating a visibility toggle menu.

The following sections develop our approach to answering these requirements.

5.2.1 Interactive Map

There are several different ways to create interactive maps on Python: Plotly, MapboxGL JS, Folium, Vega-Altair, ... Deciding which one to use came down to three criteria: the number of requirements met by the library, its ease of use, and the amount of documentation available on it; this led us to opt for Folium.

As stated previously, interactive maps are the most useful maps to display information when it is spread out over a large scale as is the case here.

The Folium[2] library allows us to visualise maps in Python through the use of the JavaScript `leaflet.js` module in the background. Creating a map is very straightforward:

```
import folium
m = folium.Map() # Generates a map
m # Displays the map
```

This code will generate a simple world map we can zoom in/zoom out on. The `Map()` function can take several parameters such as: *location*, which will generate the map focused in on the given goegraphical location; the *tiles* parameter to determine the background of the map (defaulted to OpenStreetMap, one can choose from a selection of `leaflet.js` map tiles); the *zoom_control* parameter, which allows one to add a zoom control panel to the map, it defaults to `True` and one can choose to place the panel in any of the four corners of the map.

Now that we have a map, we can add elements to it, starting with the nodes.

[2] https://python-visualization.github.io/folium/latest/index.html

5.2.2 Drawing on the Map

Nodes

To add nodes to a Folium map, one needs to use the Marker object. The procedure is as simple as the map creation: `folium.Marker(location).add_to(m)` where *location* contains the geographical coordinates of the node. The object definition can optionally take other parameters such as a *popup* which is what will appear when clicking on the marker; this is where we stored the node's name and its corresponding ID in the database in order to identify each one. One can also choose to give the marker a specific icon and/or color.

Once the marker has been created, we need to add it to the map's environment, which is done by the `add_to()` function.

Links

The links we wish to draw on the map are PolyLine objects, which are created in a very similar way as the markers: `folium.PolyLine(locations).add_to(m)`; the *locations* parameter takes a list of latitude-longitude pairs, defining the two points between which to trace a line.

These objects can take several optional parameters as well: a *popup* to display any chosen text when clicking on the line; a *tooltip* parameter which is text to be displayed when hovering over the line; and a *smooth_factor* which defines to what degree the line is simplified when zooming in.

Just as the markers, lines need to be added to the map's environment in order to appear. This is also done with the `add_to()` function.

5.2.3 Fine-tune Display

Clustering

Once all the elements we wish to see on the map have been created and added to its environment, we notice that the nodes form big unreadable clumps. To solve this issue, we can create clusters of nodes. Folium offers the possibility to do this with the MarkerCluster objects. Defining one such object and adding nodes to it might look like this:

```
import folium
m = folium.Map()
markerCluster = folium.plugins.MarkerCluster().add_to(m)
for node in nodeList:
    folium.Marker(location=(node["latitude"],
```

```
                node["longitude"]),
popup=node["name"]).add_to(markerCluster)
```

Note that we add the markers to the MarkerCluster's environment rather than the map's. If we did not change this, the markers would not group when zooming out.

Hovering over a cluster allows us to see the area it covers, and when we click on the cluster, we will automatically zoom in to display the previously clustered nodes (or smaller clusters, as we can see on Figures 5.1 through 5.3).

Figure 5.1: The clusters.

Layer Control

Layer control allows us to choose what to display on the map. It is a toggle menu in which we can select whether or not to see a certain layer; it is thus important to add elements we wish to make selectable to a FeatureGroup object. When defining one such object, we can give it several parameters: setting *overlay* to `True` means we will be able to tick/untick the group; *control* determines whether or not the group will appear in the control menu; *show* allows us to make the FeatureGroup visible on opening.

We define a FeatureGroup for the nodes, and one for each facility interconnection network which will be added to a bigger FeatureGroup containing all link layers. The layer-control we

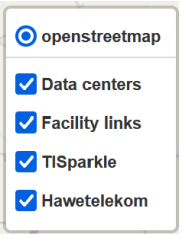


Figure 5.4: Layer-control.

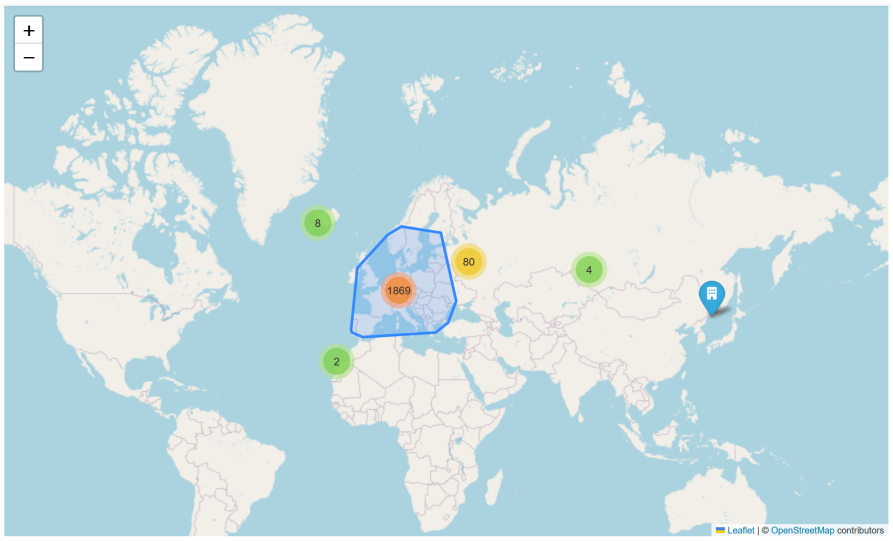


Figure 5.2: Area covered by the cluster.

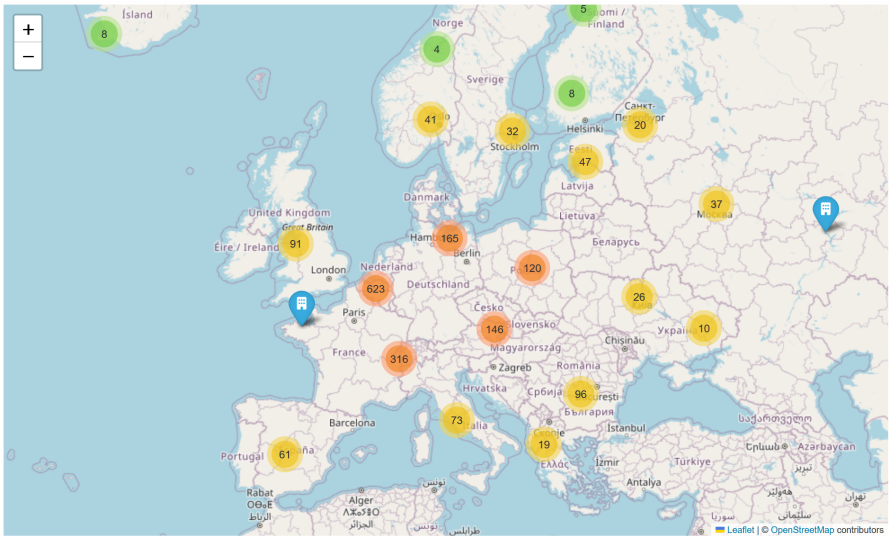


Figure 5.3: Zoom on the covered area.

obtain can be seen in Figure 5.4. In this example, we can choose to see only the links by unticking “Data centers”. “Facility links” encompasses “TISparkle” and “Hawetelekom,” and unticking one of these subgroups will only remove that group from view, whereas unselecting “Facility links” will hide both.

We execute the following code to define a FeatureGroup in order to add an element to the layer control menu, taking the data centers’ example:

```
import folium
```

```
m = folium.Map()
clusterFg = folium.FeatureGroup(name="Data centers",
                                overlay=True,
                                control=True,
                                show=True).add_to(m)
markerCluster = folium.plugins.MarkerCluster().add_to(clusterFg)
for node in nodeList:
    folium.Marker(location=(node["latitude"],
                            node["longitude"]),
                  popup=node["name"] + node["id"]).add_to(markerCluster)
```

To add the toggle menu to the map, we add `folium.LayerControl().add_to(m)` at the end, right before displaying the map (this is to ensure all elements will be added to the toggle menu).

Combining all the steps we developed in this chapter, we obtain the following snippet of code.

```
m = folium.Map() # Create map
# Clustering
clusterFg = folium.FeatureGroup(name="Data centers",
                                overlay=True,
                                control=True,
                                show=True).add_to(m)
markerCluster = folium.plugins.MarkerCluster().add_to(clusterFg)

# Place points on map
kw = {"prefix": "fa", "icon": "building"} # Define the markers' icon
for node in nodes:
    folium.Marker(location=(node["latitude"],
                            node["longitude"]),
                  popup=node["name"] + node["id"],
                  icon=folium.Icon(**kw)).add_to(markerCluster)

# Trace links in Europe
connection = folium.FeatureGroup(name="Facility links",
                                 overlay=True,
                                 control=True,
                                 show=True).add_to(m)
sub = folium.plugins.FeatureGroupSubGroup(connection,
                                          name).add_to(m)
for link in links:
    folium.PolyLine(locations=[(latitude1,longitude1),
```

```
                        (latitude2,longitude2)],
                weight,
                color,
                tooltip).add_to(sub)

folium.LayerControl().add_to(m) # Add toggle menu to map
m # Display map
```

This results in the map visible on Figure 5.5.

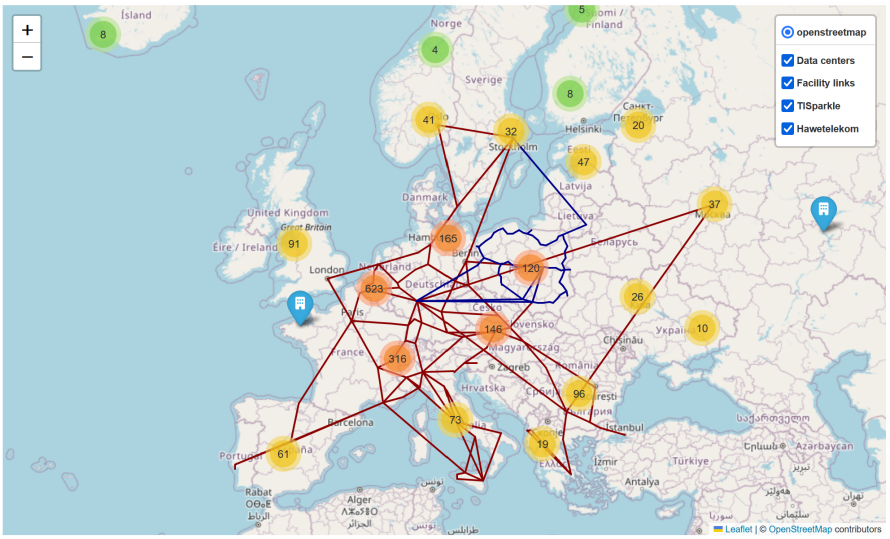


Figure 5.5: Combining all elements.

5.3 Applications

5.3.1 Shortest Path Calculation

Ultimately, the goal of this thesis was to be able to apply algorithms from graph theory, such as shortest path algorithms, to the connections we document on the map.

This was part of the reason we designed our database, allowing for the creation of a graph taking ASNs as nodes and IXes as edges. Python allows us to create a graph based on a list of nodes (with the NetworkX package), which in our case is the list of the *net* objects' IDs. To obtain this list, we launch a simple SQL `SELECT` query, the result of which we store in a list (*netPIDs*):

```
SELECT peeringID FROM net;
```

We then create adjancency lists with the help of the junction table *netix* we defined in the database, in order to establish the links and add the edges to our graph. To get these tables, we launch another SQL query:

```
SELECT ixPID, netPID FROM netix WHERE operational=1;
```

As per the definition of the *netix* table in the database, the list of resulting pairs of this query allows us to find all the networks present at an Internet Exchange (*ixHasNet*) and all the IXes a network is present at (*netInIx*). By combining this with the nodes of the graph (i.e. the networks), we can easily create the edges (two nodes are connected if and only if there exists an IX at which they are both present):

```
for n in netPIDs:
    neighbours = set()
    for ixPID in netInIx[n]:
        neighbours = neighbours | ixHasNet[ixPID]
    for neigh in neighbours:
        G.add_edge(n, neigh)
```

We then selected a series of source and target networks before applying one of NetworkX's shortest path algorithms on our graph. This results in the list of network IDs followed to arrive at the target.

To visualise this, we first need to acknowledge that a network does not have a physical presence: we cannot draw a dot on a map and say it is a network. We can, however, locate the physical presence of an IX, which we can then link to the ASes present at this location. So, in short, to visually map the shortest paths we have found, we need to reverse the nodes and edges of our graph: ASNs become edges and IXes become nodes. This is closely related to the concept of line graphs. [18]

After a few SQL manipulations to obtain the ASN associated to a network ID (which allows for more meaningful visualisation), we can generate the maps of Figures 5.6 through 5.8[3].

For one example in particular, we proceeded slightly differently: selecting two organisations with only one network (one in Iceland, the other in Malta), we were interested to see how many hops it would take to connect them. By making the assumption they are Tier 3 networks, we consider the organisations's headquarters to be the source and destination of our path on the map.

The connections visible on this map have the following meaning: Starting off in its source in Iceland, we find the network is present at an IX in London, where it connects with another AS that is present in an IX in Frankfurt, in which the Maltese network is also present, allowing us to hop to our final destination.

[3]Note, this is only a selection of shortest path candidates to illustrate the idea of how short paths are. Drawing all possible paths would have made the networks quasi-undistinguishable.

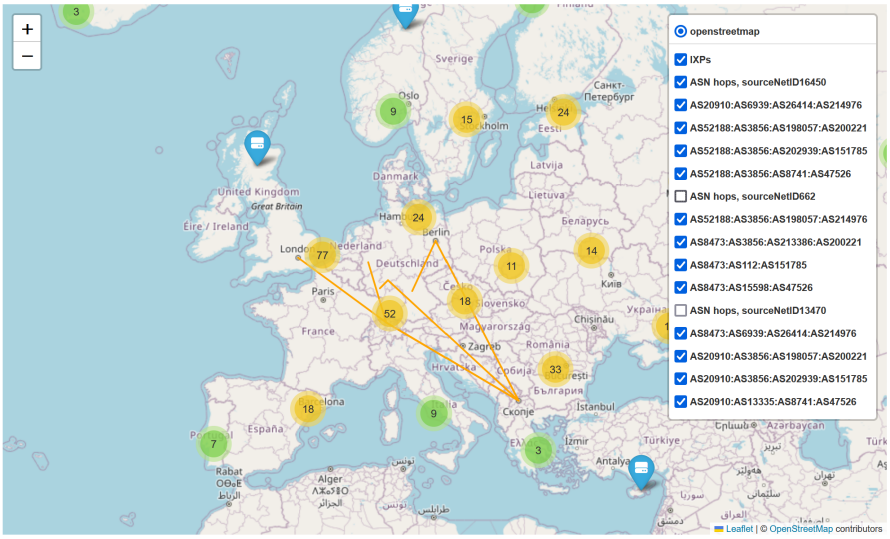


Figure 5.6: Selection of paths having AS52188 as source.

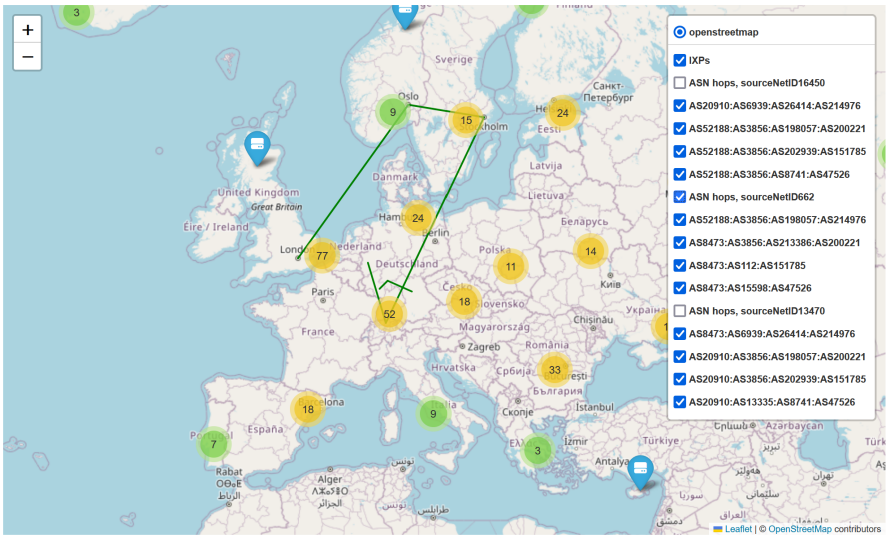


Figure 5.7: Selection of paths having AS8473 as source.

This results in the path we've traced in Figure 5.9.

5.3.2 Other Algorithms

Unweighted shortest path computation is arguably the simplest possible nontrivial graph algorithm. Our proof-of-concept is thus just that: a proof that this methodology allows one to apply graph theoretic algorithms to the data stored in our

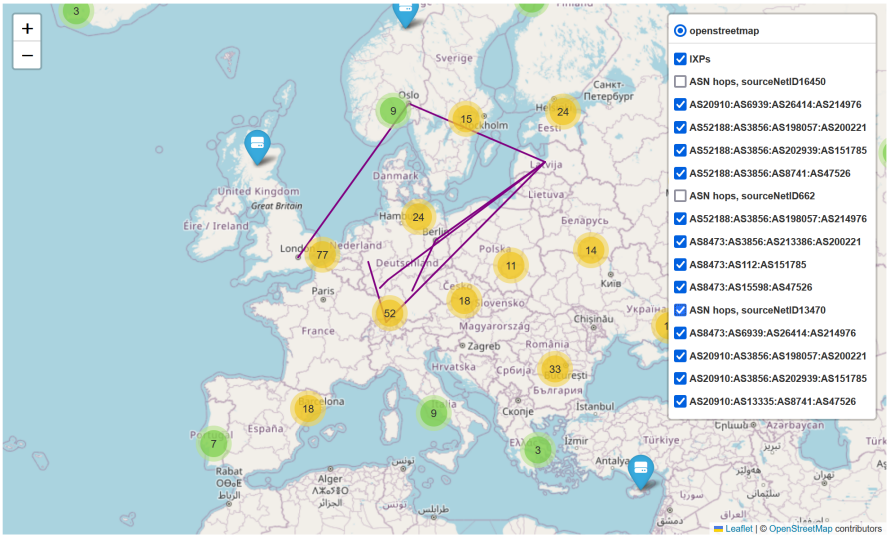


Figure 5.8: Selection of paths having AS20910 as source.

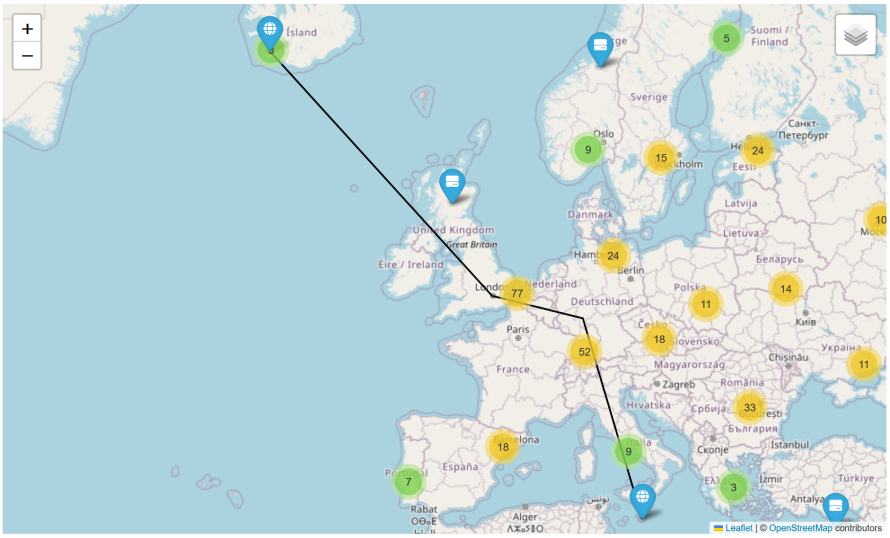


Figure 5.9: Shortest path between AS6677 and AS12709.

database. Other algorithms based on flow, redundancy, etc. can also be envisioned, although minor modifications might be necessary to register additional information about the data (for example, a suitable weighting for edges might be interesting and worth exploring).

Chapter 6

Conclusions and Future Work

In this thesis work we have looked into the problem of obtaining and consolidating comprehensive data concerning internet backbone networks spanning the European continent.

We have proposed methods to gather the data and we have shown how to extract data from various sources, how to transform them to our data model, and how to load them in a relational database.

The data in our database and the way they are stored allow to run all kinds of graph theory algorithms, e.g. shortest path calculations or path redundancy analyses. We have demonstrated this by calculating the "path of least network hops" between two randomly chosen locations in Europe.

We have also provided tooling that allows to present the calculated paths and path segments on a geographical map of the European continent.

Still, it should be noted that there are considerable gaps in the information on backbone networks available to the general public:

- in many cases we have no information on which networks, of all the networks present in an internet exchange, actually peer with each other;
- the data in PeeringDB are not guaranteed to be complete, nor free from errors;
- although in some cases we've been able to extract from operator websites data on the connections between the points-of-presence in the operator's network, these data are rather rare and often lack geographical detail.

These gaps in the information impose limitations on the accuracy of the analyses that can be conducted using the current database.

Future work should be undertaken to (try to) close these gaps. One approach can consist in replacing the current declarative data (data entered in PeeringDB at an operator's discretion, data available in maps on operator and internet exchange

websites) by 'in-situ' measurements. That is what CAIDA has set out to do with its Archipelago (Ark) Infrastructure [19]. For example, if one were able to inspect traceroute records of Internet traffic sessions initiated from many places in Europe and localise[1] the routers that intervened in the packet routing (as witnessed by the traceroute records), probably a pretty accurate image emerges of the 'paths' from AS to AS (and even 'inside' an AS). (see also [20] and [21]).

[1]Using Whois services offered by Regional Internet Registries.

Bibliography

[1] *What is an IXP?* NetNod. URL: ttps://www.netnod.se/ix/what-is-an-ixp (visited on 04/04/2024).

[2] *Advantages and Disadvantages of Network Topologies*. Inst Tools. URL: https://instrumentationtools.com/advantages-and-disadvantages-of-network-topologies/ (visited on 05/24/2024).

[3] *What is an autonomous system? What are ASNs?* CloudFlare. URL: https://www.cloudflare.com/en-gb/learning/network-layer/what-is-an-autonomous-system/ (visited on 05/24/2024).

[4] *How the International Internet Exchange Operates*. Telehouse. Oct. 13, 2015. URL: https://www.telehouse.com/how-the-international-internet-exchange-operates/ (visited on 04/03/2024).

[5] Wikipedia contributors. *Tier 1 network — Wikipedia, The Free Encyclopedia*. [Online; accessed 31-May-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Tier_1_network&oldid=1225764187.

[6] Wikipedia contributors. *Border Gateway Protocol — Wikipedia, The Free Encyclopedia*. [Online; accessed 31-May-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Border_Gateway_Protocol&oldid=1224765711.

[7] *What is BGP?* Amazon. URL: https://aws.amazon.com/what-is/border-gateway-protocol/ (visited on 04/17/2024).

[8] Kihara Kimachia. *What is BGP Routing? Understand Border Gateway Protocol*. Entreprise Networking Planet. Feb. 21, 2023. URL: https://www.enterprisenetworkingplanet.com/standards-protocols/bgp-routing/ (visited on 05/23/2024).

[9] Ben Ryall. *Introduction to PeeringDB*. PeeringDB. 2019. URL: https://docs.peeringdb.com/presentation/ngnog-pdb.pdf (visited on 04/05/2024).

[10] *PeeringDB official glossary*. URL: https://docs.peeringdb.com/glossary/ (visited on 04/05/2024).

[11] *About CAIDA*. CAIDA. URL: https://www.caida.org/about/ (visited on 04/05/2024).

[12] *About AMS-IX*. AMS-IX. URL: https://www.ams-ix.net/ams/about-ams-ix (visited on 04/23/2024).

[13] Alexander Obregon. *Object-Oriented Database vs. Relational Database — A Beginner's Guide*. Medium. Feb. 26, 2024. URL: https://medium.com/@AlexanderObregon/database-vs-relational-database-a-beginners-guide-86cc4e8357ad (visited on 05/12/2024).

[14] Tamara Pattinson. *Relational vs. Non-Relational Databases*. PluralSight. Nov. 9, 2022. URL: https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases (visited on 05/08/2024).

[15] *Les réseaux Covage: pour le déploiement de la fibre en France*. Covage. URL: https://www.covage.com/reseaux/ (visited on 04/10/2024).

[16] *Fibre Network Coverage Map*. Colt. URL: https://www.colt.net/global-network/coverage-maps/ (visited on 04/10/2024).

[17] *Getting Started with MySQL Connector/Python*. MySQL Tutorial. URL: https://www.mysqltutorial.org/python-mysql/getting-started-mysql-python-connector/ (visited on 12/20/2023).

[18] Hassler Whitney. "Congruent Graphs and the Connectivity of Graphs". In: *American Journal of Mathematics* 54.1 (1932), pp. 150–168. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2371086 (visited on 06/02/2024).

[19] *Archipelago (Ark) Measurement Infrastructure*. CAIDA. URL: https://www.caida.org/projects/ark/ (visited on 05/24/2024).

[20] *Ark IPv4 Routed /24 AS Links*. CAIDA. URL: https://catalog.caida.org/dataset/ark_ipv4_aslinks (visited on 05/24/2024).

[21] *Pushing the Boundaries with bdrmapIT: Mapping Router Ownership at Internet Scale*. CAIDA. URL: https://catalog.caida.org/paper/2018_pushing_boundaries_bdrmapit (visited on 05/24/2024).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**